On-line Coordination: Event Interaction and State Communication between Cooperative Agents

Manh Tung Pham and Kiam Tian Seow
Division of Computing Systems, School of Computer Engineering
Nanyang Technological University
Republic of Singapore 639798
Email: pham0028@ntu.edu.sg, asktseow@ntu.edu.sg

Abstract—This paper addresses a novel coordination problem for distributed agents in a discrete-event setting. We introduce and study a predicate coordination problem as the problem of distributed agents interacting and communicating between themselves to satisfy (the invariance of) a global predicate specifying an inter-agent constraint. We then develop an optimal coordination policy by which the agents can coordinate to satisfy the predicate constraint. To implement the optimal policy, we develop two on-line coordination strategies including one that can achieve significant savings in communication bandwidth, as demonstrated by simulations.

I. INTRODUCTION

Distributed multiagent coordination presents a key approach to developing complex systems. In this approach, the basic idea is to model a complex system as a network of interacting agents, and design for each agent a coordination strategy by which the individual agents can interact and communicate among themselves to manage the inter-dependencies arising due to system needs or limitations [1]. Many real-world problems such as distributed sensor nets, distributed resource allocation and distributed scheduling are important application drivers for multiagent coordination [2]. However, most of the current coordination techniques used in these complex applications are rather ad hoc and not supported by a formal design framework, and have had their performance evaluated primarily from an empirical perspective. While empirical investigation is often an inevitable process of evaluating complex applications, it is fundamentally more important to develop a generic formal basis, on which to better understand and redesign existing systems, as well as model and design new emerging systems for a variety of application domains.

In a different but related research discipline, the supervisory control of discrete-event systems (DES’s) [3], founded on the rigorous mathematical foundation of languages and automata theory [4] and partial order theory [5], is emerging as an important framework for modeling and controlling complex man-made systems. Examples include manufacturing, communication and logistics systems [4]. While supervisory control and multiagent coordination are conceptually different problems [6], interestingly, it has been shown that the mathematical concepts and algorithms from supervisory control can be re-interpreted and adapted to address the problem of coordination among discrete-event agents [6], [7], [8]. In the proposed discrete-event framework, new coordination concepts such as coordinable constraint languages [8] and optimal coordination modules [7] have been formulated as the theoretical basis for multiagent coordination. It is envisaged that the discrete-event framework, leveraging on the mathematical foundation of supervisory control, can provide a formal basis for coordination research.

Previous work [6], [7], [8] has focused on an off-line synthesis approach in which the complete coordinating actions for all anticipated interacting situations of each agent are computed off-line and stored as a coordination module. At run-time, the correct coordinating actions are then simply retrieved from the coordination module and enforced accordingly by the agent in interaction with other agents in the system. On the one hand, the off-line approach is suitable for applications where coordinating decisions have to be made as fast as possible during the agents’ run-time interaction. On the other hand, the off-line planning associated with the approach may be too expensive for applications where the number of anticipated interacting situations is too large, or when off-line planning time is limited. Against this background, this paper studies a novel coordination problem among discrete-event agents, and proposes an on-line coordination synthesis approach that complements the off-line approach proposed [6], [7], [8]. Importantly, in the on-line approach, the coordinating actions to be applied are computed by each agent only in response to situational changes. A significant merit of this on-line approach is that it avoids altogether the off-line construction of coordination modules, and therefore mitigates the inherent off-line computational complexity associated with the off-line approach. For a clear exposition to the new concepts and ideas introduced in this paper, we present and explain the theoretical results for two coordinating agents. Future work will extend the results to multiple agents.

We consider a class of discrete-event agents that can be modeled as automata (Section II), the most basic representation of DES models. Besides providing a simple yet powerful modeling formalism, automata is also amenable to composite operations and mathematical analysis [4]. Within this modeling formalism, we introduce and study a predicate coordination problem (Section III) as the problem of distributed agents interacting and communicating continually between themselves to satisfy (the invariance of) a given global predicate \( P_c \) defined on the composite state space of the agents. In essence, the predicate \( P_c \) is an inter-agent constraint of the fundamental safety type, specifying that no bad states can ever be visited during multiagent interaction. The problem is shown to be solvable in some autonomy permitted setting (Theorem 1) for coordinable predicates (Definition 3) not less restrictive than...
a given $P_c$ on the state space. The key solution developed in this paper is an optimal policy (Theorem 2) by which these agents can interact and communicate to guarantee coordination quality, in that the executing event sequences are transitions of their composite states that remain confined to the largest feasible state subset of that defined by the given predicate, and can reach all the states in this state subset (Section IV). Formally, this feasible state subset corresponds to the supremal coordinable predicate (denoted as $P_{c^\sup}$) of the given predicate $P_c$ (Definition 4).

To implement the optimal coordination policy, two cooperative agents $A_1$ and $A_2$ can coordinate as follows: Upon executing a local event or receiving a state information update, each agent would always take the action of enabling every event (defined at its current local state) provided it does not (eventually) lead the coordinated state space out of the feasible subset characterized by $P_{c^\sup}$, and disabling the event otherwise. Besides, each agent (say $A_1$) would have to decide whether or not to send its current local state to the other agent $A_2$, to provide the latter with sufficient information for computing and updating its coordinating actions. Two decision strategies are formulated in this paper:

1. In the first solution strategy, agent $A_1$ would always send its updated local state to agent $A_2$ whenever it enters a new state (Section V-A).

2. In the second, agent $A_1$ would only do so when it detects that agent $A_2$ might no longer be coordination-ready (to correctly maintain $P_{c^\sup}$) as the latter’s coordinating actions might have been invalidated (Section V-B). Agent $A_1$ detects the coordination-readiness (Definition 8) of agent $A_2$ by checking a set of local conditions for co-stability, a new coordination concept formulated in Definition 9.

Unlike the first strategy proposed which entails full communication, the second strategy, importantly, can significantly reduce the communication bandwidth, as demonstrated by experimental evaluation (Section VII), while still maintaining coordination quality (Theorem 3). As our proposed coordination problem is expressed in the rudimentary framework of predicates and finite automata, it can furnish a theoretical basis for a wide range of applications. An example (Section VI) illustrates the potential applicability of our approach, and discussion with related work (Section VIII) highlights the significance of this paper.

II. PRELIMINARIES: LANGUAGES AND AUTOMATA

A finite-state automaton $A$ is a 4-tuple $(X^A, \Sigma^A, \delta^A, x_0^A)$, where $X^A$ is the finite set of states, $\Sigma^A$ is the finite set of events, $\delta^A : \Sigma^A \times X^A \rightarrow X^A$ is the (partial) transition function, and $x_0^A \in X^A$ is the initial state. Automaton $A$ is said to be empty if its state set $X^A$ is empty. Henceforth, unless otherwise stated, an automaton is assumed to be nonempty.

Given an automaton $A$, write $\delta^A(\sigma, x)$ to denote that $\delta^A(\sigma, x)$ is defined; $\Sigma^A(x)$ to denote the set of events $\sigma \in \Sigma^A$ such that $\delta^A(\sigma, x)$; and $(\Sigma^A)^*$ to denote the set of all finite sequences (or strings) of events from $\Sigma^A$, including the empty string $\epsilon$. The definition of $\delta^A$ is extended to $(\Sigma^A)^* \times X^A$ as follows: (i) $\delta^A(\epsilon, x) = x$, and (ii) $(\forall \sigma \in \Sigma^A)(\forall s \in (\Sigma^A)^*)\delta^A(\sigma, x) = \delta^A(\sigma, \delta^A(s, x))$.

The runtime behavior of automaton $A$ can then be described by the language $L(A)$ which encompasses every string of events that can be generated from its initial state. Formally, $L(A) = \{s \in (\Sigma^A)^* : \delta^A(s, x_0^A)! \}$. For two strings $s$ and $s'$ in $(\Sigma^A)^*$, we write $s' \preceq s$ if $s'$ is a prefix of $s$, i.e., $(\exists t \in (\Sigma^A)^*)$ such that $s' = st$.

A state $x \in X^A$ is reachable if $(\exists s \in (\Sigma^A)^*) \delta^A(s, x_0^A) = x$, and automaton $A$ is reachable if all its states are reachable. If $A$ is not reachable, then a reachable automaton, denoted by $Reach(A)$, can be computed to generate the same language as $A$ by deleting from $A$ every unreachable state.

On 'equivalence' of two automata $A_1$ and $A_2$, we write $A_1 = A_2$ if they are identical in structure.

Let $A_i, i \in \{1, 2\}$, be two automata with $\Sigma^A_1 \cap \Sigma^A_2 = \emptyset$. Then their synchronous product $A_1 \parallel A_2$, models a discrete-event system of $A_1$ and $A_2$ interacting by interleaving events generated between themselves. Formally, $\Sigma^A_1 = \Sigma^A_1 \cup \Sigma^A_2$, $X^A = X^A_1 \times X^A_2$, $x_0^A = (x_0^A_1, x_0^A_2)$, and $\delta^A : \Sigma^A \times X^A \rightarrow X^A$ is given by:

$$\delta^A((\sigma, (x_1, x_2))), \text{ if } \sigma \in \Sigma^A_1(x_1);$$

$$\delta^A(\sigma, x_2), \text{ if } \sigma \in \Sigma^A_2(x_2);$$

$$\text{undefined, otherwise.}$$

III. PROBLEM FORMULATION

Consider a system of two discrete-event agents modeled by the respective reachable automata $A_1 = (X^{A_1}, \Sigma^{A_1}, \delta^{A_1}, x_0^{A_1})$ and $A_2 = (X^{A_2}, \Sigma^{A_2}, \delta^{A_2}, x_0^{A_2})$ such that $\Sigma^{A_1} \cap \Sigma^{A_2} = \emptyset$. The event set $\Sigma^{A_i}$ of agent $A_i$ is partitioned into the controllable set $\Sigma^{A_i}_c$ and the uncontrollable set $\Sigma^{A_i}_uc$. From the agent planning viewpoint, automaton $A_i$ is viewed as the local plan of the respective agent, encompassing all possible local ways to achieve the agent’s own goal; and an uncontrollable event in $\Sigma^{A_i}_uc$ is inherently autonomous and can be executed solely at the free will of the agent. As a rule, an event is pre-specified as uncontrollable if it is critical to the owner agent that disabling the event and limiting its autonomy is undesirable, expensive or even impossible.

Let $A = A_1 \parallel A_2$ model the multiagent system of $A_1$ and $A_2$ freely interacting with $\Sigma^{A_1}_c = \Sigma^{A_1}_c \cup \Sigma^{A_2}_c$ and $\Sigma^{A_1}_uc = \Sigma^{A_1}_uc \cup \Sigma^{A_2}_uc$. The two agents $A_1$ and $A_2$ would need to coordinate between themselves if, due to system needs or limitations, the execution of some event sequences in $L(A)$ is undesirable and must be prevented. In other words, their coordinating actions would need to satisfy an inter-agent constraint that excludes undesirable event sequences.

Consider an inter-agent constraint specified by an automaton $C$ (representing the language $L(C)$). Then the coordination problem becomes that of $A_1$ and $A_2$ interacting and communicating to conform to $C$, such that none of the sequences in the bad sequence set $L(A) - L(C)$ can ever be generated during multiagent interaction. The automaton $C$ is essentially an inter-agent constraint of the safety type, specifying that nothing bad can happen.

In this paper, we focus on a $C$ which is a nonempty sub-automaton of $A$, i.e., $X^C \subseteq X^A$, $x_0^C = x_0^A$, and $\delta^C$ is a restriction of $\delta^A$ on $\Sigma^A \times X^C$. Such an automaton $C$ can be equivalently represented by a predicate defined on the set $X^A$. In essence, a predicate $P$ defined on $X^A$ is a function $P : X^A \rightarrow \{0, 1\}$. For automaton $C$, the equivalent predicate
$P_c$ is defined as follows:

$$(\forall x \in X^A) P_c(x) = \begin{cases} 1, & \text{if } x \in X^C; \\ 0, & \text{otherwise}. \end{cases}$$

Henceforth, such a constraint automaton $C$ and its equivalent predicate $P_c$ can be used interchangeably. For a state $x \in X^A$, we say $x$ satisfies $P_c$, and write $x \models P_c$ if $P_c(x) = 1$. For two predicates $P_1$ and $P_2$ defined on $X^A$, we say that $P_1$ is not less restrictive than $P_2$, denoted by $P_1 \preceq P_2$, if $(\forall x \in X^A) (x \models P_1 \Rightarrow x \models P_2)$.

In addressing what can now be called a predicate coordination problem, $P_c$ is an inter-agent constraint of the fundamental safety type, specifying that no states in the bad state set $X^A - X^C$ can ever be visited during multiagent interaction.

The coordinating actions for a pair of agents are governed by a coordination policy, formally defined as follows.

**Definition 1.** A coordination policy $\pi_{A_1, A_2}$ is a pair of agent policies $< \pi_{A_1}, \pi_{A_2} >$, where $\pi_{A_i}$ for agent $A_i$ is a mapping from a state $x \in X^A$ to an event subset of $\Sigma^A$, such that $(\forall x \in X^A) \pi_{A_i}(x) \subseteq \Sigma^A_{uc} \cap \Sigma^A(x)$.

Thus $\pi_{A_i}$ attaches to each state $x$ of $X^A$ a subset of $\Sigma^A_i$ that contains $\Sigma^A_{uc} \cap \Sigma^A(x)$ - the subset of uncontrollable events of $A_i$ that are defined at $x$. Using coordination policy $\pi_{A_1, A_2}$, agent $A_i$, upon observing the system state $x \in X^A$, enables every event $\sigma \in \pi_{A_i}(x)$, and disables all other events. The condition $(\forall x \in X^A) \pi_{A_i}(x) \supseteq \Sigma^A_{uc} \cap \Sigma^A(x)$ characterizes the fact that uncontrollable events can never be disabled.

**Definition 2.** The system of agents $A_1$ and $A_2$ interacting using a coordination policy $\pi_{A_1, A_2}$ is a (discrete-event) system represented by an automaton $A_\pi = \text{Reach}(X^A, \Sigma^A, \delta_\pi, x_0^A)$, where $(\forall \sigma \in \Sigma^A_i) (\forall x \in X^A) \delta_\pi^A(\sigma, x) = \delta^A(\sigma, x)$ if $\delta^A(\sigma, x)$! and $\sigma \in \pi_{A_i}(x)$, and is undefined otherwise.

Since we are only interested in the reachable part of the coordinated system, the condition $(\forall x \in X^A) \pi_{A_i}(x) \supseteq \Sigma^A_{uc} \cap \Sigma^A(x)$ of a coordination policy $\pi_{A_1, A_2}$ in Definition 1 can be relaxed to $(\forall x \in X^A) \pi_{A_i}(x) \supseteq \Sigma^A_{uc} \cap \Sigma^A(x)$.

**Definition 3.** A predicate $P$ defined on $X^A$ is said to be coordinate if, for every $x \in X^A$, satisfing $P$, the following conditions are satisfied: (1) $(\exists s \in (\Sigma^A)^*) [\delta^A(s, x_0^A) = x$ and $(\forall u \leq s) \delta^A(u, x_0^A) \models P]$, and (2) $(\forall \sigma \in \Sigma^A) [\delta^A(\sigma, x) \Rightarrow \delta^A(\sigma, x)]$.

By Definition 3, coordinability asserts that if $x$ satisfies $P$ then (i) $x$ is reachable from $x_0^A$ via a sequence of states satisfying $P$ [Condition (1)], and (ii) if $\sigma \in \Sigma^A_{uc}$ and $\delta^A(\sigma, x)!$, then $\delta^A(\sigma, x)$ satisfies $P$ [Condition (2)].

**Theorem 1.** Let $C$ be a nonempty sub-automaton of $A$. Then there exists a coordination policy $\pi_{A_1, A_2}$ such that $A_\pi = C$ if and only if $P_C$ is coordinable.

**Proof:** Let $\pi_{A_1, A_2}$ be a coordination policy with each agent policy $\pi_{A_i}$ given as: $(\forall x \in X^A) \pi_{A_i}(x) = \Sigma^A(x) \cap \Sigma^A_i$. Since $P_c$ is coordinable, by Condition (2) of Definition 3, $(\forall x \in X^A) \pi_{A_i}(x) \supseteq \Sigma^A_{uc} \cap \Sigma^A(x)$. Moreover, by Condition (1) of Definition 3, $C$ is a reachable automaton. Hence it follows that $A_\pi = C$.

(Only If) Let $\pi_{A_1, A_2}$ be a coordination policy with $A_\pi = C$. Since $A_\pi$ is a reachable automaton, $P_c$ trivially satisfies Condition (1) of Definition 3. And, since $(\forall x \in X^A) \pi_{A_i}(x) \supseteq \Sigma^A_{uc} \cap \Sigma^A(x)$, it follows that $P_c$ satisfies Condition (2) of Definition 3. Hence $P_c$ is coordinable. ■

It can be shown that the set of coordinable predicates that are not less restrictive than $P_c$ is nonempty and closed under arbitrary predicate disjunctions, and so its supremal element exists.

Supremal coordinable predicate $P_c^{\text{sup}}$ of a given predicate $P_c$ is defined as follows.

**Definition 4.** Given a predicate $P_c$ defined on the system state space $X^A$. The supremal coordinable predicate of $P_c$, denoted by $P_c^{\text{sup}}$, is the unique predicate defined on $X^A$ which satisfies the following properties: (1) $P_c^{\text{sup}} \preceq P_c$, (2) $P_c^{\text{sup}}$ is coordinable, and (3) $(\forall P \preceq P_c) [\text{if } (P \text{ is coordinable)} \Rightarrow (P \preceq P_c^{\text{sup}})]$.

Let $C^{\text{sup}}$ denote the equivalent automaton of $P_c^{\text{sup}}$. The predicate coordination problem can now be formally stated as follows.

**Problem 1.** Given a predicate constraint $P_c$ defined on the system state space $X^A$, construct the (unique) optimal coordination policy $\pi_{A_1, A_2}$ such that $A_\pi = C^{\text{sup}}$.

The solution policy for Problem 1 is said to be optimal since the agents implementing it do not disable their own controllable events unless doing so may eventually lead the coordinated state space out of the state subset satisfying $P_c$. Therefore, the policy enables the agents to visit as many states in $X^C$ as possible, and have maximal autonomy over their own actions.

Note that (i) $C^{\text{sup}} = C$ if $P_c$ is coordinable, and (ii) $P_c^{\text{sup}}$ can be a false predicate, i.e., $(\forall x \in X^A) P_c^{\text{sup}}(x) = 0$; in this case, $C^{\text{sup}}$ is an empty automaton, and Problem 1 has no solution.

**IV. COORDINATION SYNTHESIS**

**Definition 5.** A state $x \in X^A$ is said to be $P_c$-safe if $(\forall s \in (\Sigma^A_{uc})^*) [\delta^A(s, x) \Rightarrow \delta^A(s, x) \models P]$.

Thus a state $x$ is $P_c$-safe if every state reachable from $x$ via a string of uncontrollable events satisfies $P_c$. Following, it can be shown that $C^{\text{sup}}$ is an empty automaton if and only if the initial state state $x_0^A$ is $P_c$-unsafe. In synthesizing the solution for Problem 1, we shall henceforth assume that $x_0^A$ is $P_c$-safe.

Given a state $x \in X^A$, the following *CheckSafety* procedure returns true if $x$ is $P_c$-safe, and false otherwise.

**Procedure  CheckSafety (x ∈ X^A)**

\[
\text{begin if Safety}[x] \neq \text{NIL then Return Safety}[x]; \\
\text{Return BFS}[\text{Checking}(x)]; \\
\text{end}
\]

*CheckSafety* determines the safety value of a given system state $x$ by performing a search over part of or the entire system model $A = A_1 || A_2$, and possibly also using stored results from prior (step) computations. It maintains a global logic variable $\text{Safety}$ to store the safety value of every state in
$X^A$. $Safety[x]$ is true if $x$ is $P_s$-safe, false if $x$ is $P_s$-unsafe, and NIL if the safety value of $x$ has not been determined. If $Safety[x] \neq NIL$, i.e., the safety value of $x$ has been computed in prior computations, the procedure simply returns the stored value. Otherwise, it invokes a procedure called $BFSChecking$ to determine the safety value of $x$.

**Procedure $BFSChecking(x \in X^A)$**

```plaintext
begin
  Expanded $\leftarrow \{\};$ $Q \leftarrow \{x\};$ $father[x] \leftarrow NIL;$
  while $Q \neq \emptyset$ do
    $u \leftarrow$ the head element of $Q;$ $Q \leftarrow Q \cup \{u\};$
    if $u \in Expanded$ then continue;
    Expanded $\leftarrow Expanded \cup \{u\};$
    foreach $\sigma \in \Sigma^A(u) \cap \Sigma^A_{uc}$ do
      $v \leftarrow \delta^A(\sigma, u);$ if $Safety[v] == false or v $\not\in P_s$ then
        if $\neg Safety[v]$ or $v \notin father[v];$
        $Return \ false;$
        if $Safety[v] == false$ or $v \notin father[v] ;$
        $Return \ false;$
      fi
    fi
  fi
  foreach $u \in Expanded$ do
    $Safety[u] \leftarrow true;$
  Return $true;$
end
```

$BFSChecking$ builds a $\Sigma^A_{uc}$-tree rooted at $x$ by expanding all the consecutive $\Sigma^A_{uc}$-successors of $x$ in a breadth-first fashion. It maintains several data structures for this expansion process: a first-in, first-out queue to manage the set of states to be expanded next, a queue $Expanded$ to store every state that has already been expanded, and, for every expanded state $u$, a variable $father[u]$ to store its predecessor. By Definition 5, $BFSChecking$ determines the safety value of $x$ as follows. If, during the expansion process, there is some $\Sigma^A_{uc}$-successor $v$ of $x$ that does not satisfy $P_s$, or has already been shown to be $P_s$-unsafe in prior computations, then $x$ and all of its $\Sigma^A_{uc}$-successors in the path leading $x$ to $v$ are $P_s$-unsafe. In this case, the procedure simply stores the $P_s$-unsafe values of these states in variable $Safety$ and returns false. Otherwise, if the $\Sigma^A$-tree rooted at $x$ is expanded fully without encountering any $P_s$-unsafe state, then $x$ and all its consecutive $\Sigma^A_{uc}$-successors are $P_s$-safe. Thus, the procedure simply stores the $P_s$-safe values of every expanded state $u \in Expanded$ in variable $Safety$, and returns true. Using a breadth-first search, procedure $CheckSafety$ has worst-case time complexity which is linear in the total number of states and uncontrollable transitions of its input automaton $A$. Thus the highest complexity bound is linear in the (state plus transition) size of $A$. In an upper bound, the complexity is $O(|X^A| + |X^A| \times |\Sigma^A_{uc}|)$ or $O(|X^A| \times |\Sigma^A_{uc}|)$.

**Theorem 2.** Let $C$ be a nonempty sub-automaton of $A$. Assume that $x^0_0$ is $P_s$-safe. Let $\pi_{<A_1,A_2>}$ be a coordination policy with each agent policy $\pi_{A_i}$ given as: $\forall x \in X^A \forall \sigma \in \Sigma^A_i \exists \pi_{A_i}(x)$ if and only if $\delta^A_i(\sigma, x)$ is $P_s$-safe. Then $A = C^{sup}$ (i.e., $\pi_{<A_1,A_2>}$ is the optimal solution policy of Problem 1).

**Proof:** Since $x^0_0$ is $P_s$-safe, $(\forall x \in X^A) \pi_{A_i}(x) \supseteq \Sigma^A_i(x) \cap \Sigma^A_{uc}$. Let $P_{A_s}$ be an equivalent predicate of $A_s$. Trivially, $P_{A_s} \leq P_s$. We now have to show that (i) $P_{A_s}$ is coordinate, and (ii) for any coordinate predicate $P \preceq P_{A_s}$, $P \leq P_{A_s}$.

(i) Since $A = A_s$ is a reachable automaton, $P_{A_s}$ trivially satisfies Condition (1) of Definition 3. Moreover, since $(\forall x \in X^A)(x \in \Sigma_{uc} \delta^A_i(\sigma, x) = P_s$-safe, implying $\delta^A_i(\sigma, x) = P_s$. It follows that $P_{A_s}$ satisfies Condition (2) of Definition 3. Hence $P_{A_s}$ is coordinate.

(ii) Let $P \preceq P_s$ be a coordinate predicate. Let $x \in X^A$ be an arbitrary state that satisfies $P$. Then $x$ is reachable from $x^0_0$ via a sequence of states satisfying $P$, and $(\forall x \in X^A(\bigwedge x \in \Sigma_{uc} \delta^A_i(\sigma, x) = P)$. For a string $s \in \Sigma_{uc}^*$, by induction on the length of $s$, we infer that if $\delta^A(s, x) = P_s$, then $\delta^A(s, x) = P$. Hence, since $P \preceq P_s$, $(\forall x \in \Sigma_{uc}^*) \delta^A(s, x) = P_s$, which in turn implies that $x$ is $P_s$-safe, or $x = P_{A_s}$. Hence $P \preceq P_{A_s}$.

By Theorem 2, to implement the optimal solution policy of Problem 1, the following $ComputeEnabledEventSet$ procedure could be used by agent $A_i (i \in \{1, 2\})$ to determine the set of events to enable next each time it observes a new state system $x$.

**Procedure $ComputeEnabledEventSet(x \in X^A)$**

```plaintext
begin
  $\pi_{A_i}(x) \leftarrow \Sigma^A_i(x) \cap \Sigma^A_{uc};$
  foreach $\sigma \in \Sigma^A_i(x) \cap \Sigma^A_{uc}$ do
    if $CheckSafety(\delta^A_i(\sigma, x) == true then$
      $\pi_{A_i}(x) \leftarrow \pi_{A_i}(x) \cup \{\sigma\}$
    fi
  fi
  $Return \pi_{A_i}(x);$;
end
```

The procedure has worst-case time complexity of $O(|\Sigma^A_i| \times |\Sigma^A_{uc}|)$ since it has to invoke procedure $CheckSafety$ exactly $|\Sigma^A_i| \times |\Sigma^A_{uc}|$ times.

V. ON-LINE COORDINATION STRATEGIES

To implement the optimal coordination policy given in Theorem 2, i.e., policy $\pi_{<A_1,A_2>}$ for which $A = C^{sup}$, we now present two on-line coordination strategies that enable the agents to interact and individually compute their next coordinating actions in response to continual situational changes.

A. With Full Communication

The first on-line strategy is called OnlineCoAgent-ComFull (Fig.1), and follows directly from Theorem 2. Using the strategy, the agents start by exchanging their initial states, and upon entering a new state, an agent would immediately send its updated local state to the other agent. Each time the agents have individually updated the system state, they would apply procedure $ComputeEnabledEventSet$ to determine their next set of events to enable.

Although easy to implement, OnlineCoAgent-ComFull entails full communication and therefore may not be desirable for situations in which communication bandwidth is a scarce resource. For such situations, other coordination strategies which could reduce communication are needed. OnlineCoAgent-ComFull can however be used to benchmark against the effectiveness in bandwidth reduction of these strategies.
OnlineCoAgent-ComFull($A_i$)

\begin{verbatim}
begin
  Communicate the initial state $x_{0i}^A$ to $A_2$;
end

Upon receiving local state $x_2$ from $A_2$
\begin{verbatim}
begin
  Update system state $x \leftarrow (x_1, x_2)$;
  Apply ComputeEnabledEventSet($x$) to determine the next set of events to enable;
end

Upon executing event $\sigma \in \Sigma^{A_1}$
\begin{verbatim}
if $x_1 \not\equiv \delta^{A_1}(\sigma, x_1)$ then
  Update system state $x \leftarrow (\delta^{A_1}(\sigma, x_1), x_2)$;
  Communicate the current local state $x_1$ to $A_2$;
  Apply ComputeEnabledEventSet($x$) to determine the next set of events to enable;
end
\end{verbatim}
\end{verbatim}

Fig. 1: OnlineCoAgent-ComFull($A_i$)($i \in \{1, 2\}$). On-line coordination strategy with full communication for agent $A_i$. For definiteness of description, the strategy instance for $A_1$ is shown; that for $A_2$ is the same except that its reciprocal agent is $A_1$.

B. With Reduced Communication

The second coordination strategy attempts to reduce communication bandwidth. It uses the concept of an agent's (coordination) view to implement the optimal policy given in Theorem 2.

The local view of agent $A_1$ is represented by the tuple $(x_1, x_{1}^{x_1}, x_{2}^{x_1})$, where $x_1$ is its current state, $x_{1}^{x_1}$ is $A_1$’s view of $A_2$’s current state and is the most recent state information $A_1$ received from agent $A_2$, and $x_{2}^{x_1}$ is the most recent state information that $A_1$ sent to $A_2$. The local view of agent $A_2$ is similarly represented by the tuple $(x_2, x_{1}^{x_2}, x_{2}^{x_2})$. Note that since inter-agent communication is assumed instantaneous, $x_{1}^{x_2}$ and $x_{2}^{x_2}$ might be different from $x_1$ and $x_2$, respectively. When $x_{1}^{x_2} = x_1$ and $x_{2}^{x_1} = x_2$, i.e., both the agents have the most current information about each other’s local state, the agents are said to be totally synchronized.

To always achieve total synchronization, the agents would have to send their updated local state to the other whenever they enter a new state, i.e., they would have to follow the full communication approach. Since total synchronization may not always be necessary for coordination, the communication needs between coordinating agents could be reduced by enabling the agents to strategically decide, based on their local view, whether or not to send their current local state to the other. In developing one such strategy, the following definition is needed.

Definition 6. Given $x_1 \in X^{A_1}$, two states $x_2, x_2' \in X^{A_2}$ are said to be equivalent with respect to $x_1$ (on $P_c$), and denoted by $x_2 \equiv x_1 x_2'$ (mod $P_c$), if $(\forall \sigma \in \Sigma^{A_1}(x_1) \cap \Sigma^{A_2}) (\delta^{A_1}(\sigma, x_1), x_2) \in P_c$-safe if and only if $(\delta^{A_1}(\sigma, x_1), x_2') \in P_c$-safe. The notion $x_1 \equiv x_1 x_2'^{A_2}$ is defined similarly.

Intuitively, for $i, j \in \{1, 2\}$, $x_i \equiv x_j x_i'$ (mod $P_c$) means that whether agent $A_i$ is in state $x_i$ or $x_i'$, the coordinating actions of agent $A_j$ (regarding which events it should enable or disable) are the same when maintaining $P_c$. An important implication is that if $A_j$ is in state $x_j$, $A_i$ does not need to inform $A_j$ when it moves from state $x_i$ to $x_i'$.

For economy of notation, we will often omit ‘mod $P_c$’ and simply write $x_i \equiv x_j x_i'$ in place of $x_i \equiv x_j x_i'$ (mod $P_c$) when no ambiguity can arise.

Given $x_j \in X^{A_j}$, $\equiv x_j$ defines an equivalence relation on the state set $X^{A_j}$. As per usual, a partial order relation can be defined over the set of those equivalence relations.

Definition 7. For two states $x_1, x_1' \in X^{A_1}$, $\equiv x_1'$ is said to be finer than $\equiv x_1$, and denoted by $\equiv x_1 \preceq \equiv x_1'$, if $\forall x_2, x_2' \in X^{A_2}(x_2 \equiv x_1 x_2' \Rightarrow (x_2 \equiv x_1' x_2')$. The notion $\equiv x_2 \preceq \equiv x_2'$ for $x_2, x_2' \in X^{A_2}$ is defined similarly.

Thus, $\equiv x_j \preceq \equiv x_j'$ means that if $A_j$ does not need to inform $A_i$ in state $x_i$ when it moves from state $x_i$ to $x_i'$, it also does not need to do so if $A_j$ is in state $x_j'$.

We can now define the main concept called coordination-readiness that characterizes when the two coordinating agents $A_1$ and $A_2$ can correctly determine their next set of enabled events to maintain $P_c$.

Definition 8. Two agents $A_1$ and $A_2$, with their respective local views $(x_1, x_{1}^{x_1}, x_{2}^{x_1})$ and $(x_2, x_{1}^{x_2}, x_{2}^{x_2})$, are said to be coordination-ready (for $P_c$) if $x_1^{x_1} \equiv x_1 x_2$ and $x_2^{x_2} \equiv x_2 x_1$.

Thus, the two agents are coordination-ready if, $x_{1}^{x_1}$, the most recent state information $A_1$ sent to $A_2$, presents $A_2$ with the equivalent next-state information associated with $x_1$, the current state of $A_1$, for determining the same $P_c$-safety value of every next system state that can result from $A_2$’s execution of a controllable event from its current state $x_2$, $i, j \in \{1, 2\}$.

Hence, to implement the optimal solution policy $\pi_{A_1, A_2}$ for which $A_2 = C^{opt}$ (Theorem 2), the agents, following every event execution, would need to re-establish coordination-readiness prior to determining their next set of enabled events. Note that always re-establishing total synchronization as with OnlineCoAgent-ComFull is the most conservative way that trivially and implicitly re-establishes coordination-readiness. Checking for coordination-readiness first, with $x_{1}^{x_1} \equiv x_1 x_2$ by agent $A_1$, might reduce $A_2$ to communicating its current local state to the other agent $A_2$ only when the check fails. However, such direct checking clearly requires agent $A_1$ to also know the current local state $x_j$ of agent $A_j$, which is not always possible. This necessitates a stronger notion called co-stability, whose conditions can be mutually checked by the agents.

Definition 9. Two agents $A_1$ and $A_2$ with their respective local views $(x_1, x_{1}^{x_1}, x_{2}^{x_1})$ and $(x_2, x_{1}^{x_2}, x_{2}^{x_2})$, are said to be co-stable (for $P_c$) if (1) $x_1^{x_1} \equiv x_1 x_1$, (2) $x_2^{x_2} \equiv x_2 x_2$, (3) $\equiv x_1 \preceq \equiv x_2$, and (4) $\equiv x_2 \preceq \equiv x_1$.

The following proposition formally states that co-stability is a sufficient condition for coordination-readiness.
Proposition 1. Whenever agents $A_1$ and $A_2$ are co-stable (for $P_\sigma$), they are coordination-ready (for $P_\sigma$).

Importantly, the co-stability conditions could be mutually checked by the two agents $A_1$ and $A_2$ as follows. Conditions (1) and (3), which only require information access to $x_1^1$, $x_2^1$, and $x_1^2$, can be checked by agent $A_1$ using its local view $(x_1^1, x_2^1, x_1^2)$. To check Condition (1), i.e., whether $x_1^2 \equiv x_1^2 \equiv x_2^1$, $A_1$ can simply check, for each $\sigma \in \Sigma_{A_2}(x_1^1) \cap \Sigma_{A_1}$, whether the two system states $(x_1^1, \delta_{A_2}(\sigma, x_2^1))$ and $(x_1^1, \delta_{A_1}(\sigma, x_2^1))$ have the same $P_\sigma$-safety value. This checking process has worst-case time complexity of $O(|\Sigma_{A_2}(x_1^1) \times |\Sigma_{A_1}|)$ since it involves invoking procedure \textit{CheckSafety} exactly $2 \times |\Sigma_{A_2}(x_1^1) \cap \Sigma_{A_1}|$ times. Similarly, to check Condition (3), i.e., whether $\equiv x_1^2 \equiv x_2^1$, $A_1$ can iterate over every state pair $x_2^1, x_2^2 \in X_{A_2} \times X_{A_2}$ and check whether $(x_2^1 \equiv x_2^2 \equiv x_2^1)$ implies $(x_2^1 \equiv x_2^1 \equiv x_2^2)$. This checking process can be shown to have worst-case time complexity of $O(|\Sigma_{A_2}(x_1^1) \times |\Sigma_{A_1}|)$. Conditions (2) and (4) can be checked by agent $A_2$ in a similar manner.

Thus, to re-establish coordination-readiness following an event execution, the agents can check the co-stability conditions, and when necessary, interact by communicating their local state to re-establish co-stability using a new strategy called \textit{OnlineCoAgent-ComReduce} (Fig.2); we call this process co-stabilization.

\begin{algorithm}
\caption{OnlineCoAgent-ComReduce($A_1$)}
\begin{algorithmic}
\State Communicate the initial state $x_0^1$ to $A_2$;
\State Upon receiving local state $x_2^1$ from $A_2$
\begin{algorithmic}
\State \textbf{begin}
\State \quad Update the view of $A_2$’s state: $x_2^1 \leftarrow x_2^1$;
\State \quad \textbf{if} $x_1^1 \neq x_2^1 \equiv x_1^1 \then$
\State \quad \quad Communicate $x_1^1$ to $A_2$; Update $x_1^2 \leftarrow x_1^1$;
\State \quad \quad \textbf{Apply} ComputeEnabledEventSet($x_1^1, x_1^2$))
\State \quad \textbf{end}
\State \textbf{end}
\State Upon executing event $\sigma \in \Sigma^{A_1}$
\begin{algorithmic}
\State \textbf{begin}
\State \quad Update current state: $x_1^1 \leftarrow \delta^{A_1}(\sigma, x_1^1)$; \textbf{if} $\equiv x_1^2 \not\equiv x_1 \or x_1^1 \equiv x_1 \then$
\State \quad \quad Communicate $x_1^1$ to $A_2$; Update $x_1^2 \leftarrow x_1^1$;
\State \quad \quad \textbf{Apply} ComputeEnabledEventSet($x_1^1, x_1^2$))
\State \quad \textbf{end}
\end{algorithmic}
\end{algorithmic}
\end{algorithm}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig2.png}
\caption{OnlineCoAgent-ComReduce($A_1$) ($i \in \{1, 2\}$) - On-line coordination strategy with reduced communication for agent $A_i$. For definiteness of description, the strategy instance for $A_1$ is shown; that for $A_2$ is the same except that its reciprocal agent is $A_1$.}
\end{figure}

Note that the strategy OnlineCoAgent-ComFull (Fig.1) implicitly and trivially guarantees co-stability, by requiring the two agents to always communicate to re-establish total synchronization between themselves. As Theorem 3 below formally states, the strategy OnlineCoAgent-ComReduce (Fig.2) can also attain co-stability, but without the agents always having to achieve total synchronization. Importantly, this suggests that the latter strategy can reduce inter-agent communication.

Theorem 3. Using OnlineCoAgent-ComReduce, two agents $A_1$ and $A_2$ can always, after every event execution, co-stabilize between themselves (and hence become coordination-ready) for $P_\sigma$.

\textbf{Proof}: Using OnlineCoAgent-ComReduce, the agents start by exchanging their initial local states, and are initially co-stable. Now, assume that the agents are currently co-stable with their respective local views $(x_1^1, x_2^1, x_1^2)$ and $(x_2^1, x_1^2, x_2^1)$. Upon executing a local event $\sigma$ and moving to a new local state, using OnlineCoAgent-ComReduce, one agent (say $A_1$) updates its local state to $x_1^1$ and initiates the communication process with the other agent $A_2$ as follows.

- $A_1$ first checks if the two conditions $\equiv x_1^2 \equiv x_1^1$ and $x_1^2 \equiv x_1^1$, $x_1^1$ are satisfied. If so, the agents are already co-stable, and no inter-agent communication is needed.
- If, however, $\equiv x_1^2 \equiv x_1^1$ or $x_1^1 \equiv x_1^2$, $x_1^1$, $A_1$ communicates $x_1^1$ to $A_2$, and updates its local view to $(x_1^2, x_1^2, x_1^1)$, validating Condition (1) $(x_1^1 \equiv x_1^2 \equiv x_1^1)$ and Condition (3) $(\equiv x_1^2 \equiv x_1^1)$ of co-stability. Upon receiving $x_1^1$, $A_2$ updates its local view to $(x_2^2, x_1^1, x_2^2)$. Condition (4) $(\equiv x_1^1 \equiv x_1^2)$ is still satisfied because the agents are co-stable prior to $A_1$ executing event $\sigma$. $A_2$ then proceeds to check Condition (2) $(x_1^1 \equiv x_1^2)$. The cases are two.

- \textit{Case (1)}: $\equiv x_1^2 \equiv x_1^1$. Together with $x_1^2 \equiv x_2^1$, $x_2$ (because the agents are co-stable prior to $A_1$ executing event $\sigma$), it implies $x_1^2 \equiv x_1^2$, i.e., Condition (2) is satisfied. In this case, the agents are co-stable after $A_2$ has acted upon the communication message $x_1^1$ received from $A_1$, and thus no further communication is needed.
- \textit{Case (2)}: $\equiv x_1^2 \equiv x_1^1$. In this case, Condition (2) may not be satisfied. If so, $A_2$ then communicates $x_2$ to $A_1$, and in response, $A_1$ updates its view of $A_2$’s current state from $x_1^1$ to $x_2$. Now the two agents are totally synchronized and are therefore co-stable.

Thus, the two agents can always, after every event execution, co-stabilize between themselves.

Thus, OnlineCoAgent-ComReduce enables two coordinating agents $A_1$ and $A_2$ to maintain the supremal coordinable predicate of the constraint $P_\sigma$, and hence implement the optimal solution policy of Problem 1.

VI. ILLUSTRATIVE EXAMPLE

We now present an example to explain the effectiveness of our on-line coordination strategy with reduced communication. The example under study is an exploration problem consisting of two agents $A_1$ and $A_2$ concurrently exploring a common space. The common space to be explored consists of three regions $A$, $B$ and $C$ which are far away from each other, and each region consists of $n$ rooms to be explored (Fig.3). To explore the space, each agent moves from one room to another, and explores each room individually. Since the regions are far from each other, the cost of moving from one region to another is much more expensive than the cost of moving among the rooms in the same region. Therefore, after entering one region, the agents would want to freely explore every
room in the region before moving to another region. Thus it is reasonable to pre-specify the events representing each agent moving inside a region as uncontrollable, and those representing each agent moving from one region to another as controllable.

Consider an inter-agent (predicate) constraint $P_c$ specifying that ‘the two agents must not explore the same room at the same time’. Since an agent could uncontrollably move from one room to another in the same region, it is easy to see that the supremal coordinate predicate of $P_c$, denoted by $P_{c\sup}$, is that ‘the two agents must not explore the same region at the same time’. Agents $A_1$ and $A_2$ can utilize the proposed OnlineCoAgent-ComReduce strategy to interact and communicate to maintain $P_{c\sup}$ as follows.

Defining only those states necessary for illustration, for $1 \leq i \leq n$, let $a_{1,i}$ denote the state of agent $A_1$ when it is exploring room $i$ in region $A$; and $b_{2,i}$ and $c_{2,i}$ denote the states of agent $A_2$ when it is exploring room $i$ in region $B$ and region $C$ respectively. From Definitions 6 and 7, it can be verified that: $(\forall i, j, k, h \in [1, n]) \ b_{2,i} \equiv_{a, k} \ b_{2,j}, \ c_{2,i} \equiv_{a, k} \ c_{2,j}, \equiv_{a, i} \equiv_{a, k} \ c_{2,j}, \equiv_{a, i} \equiv_{a, k} \ c_{2,i}$ and $\equiv_{b, j} \equiv_{b, i}$ (1).

Suppose $A_1$ is exploring room $k$ in region $A$ and $A_2$ is exploring room $i$ in region $B$, and their respective local views are $(a_{1,i}, b_{2,j}^1, a_{1,k}^2)$ and $(b_{2,i}^1, a_{1,i}^2, b_{1,j}^2)$, where $b_{2,i}$ and $a_{1,i}$ might be different from $b_{2,i}$ and $a_{1,k}$. By (1), the agents are co-stable. Now, suppose agent $A_2$ moves to another room. Whether $A_2$ needs to inform $A_1$ of its updated local state will depend on whether the agents need to re-establish co-stability:

1. If $A_2$ moves to room $r$ in region $B$, its local view becomes $(b_{2,r}^1, a_{1,r}^2, b_{2,j}^2)$. By (1), $A_2$ can locally verify that the agents are still co-stable, and therefore does not need to communicate its updated local state to $A_1$.

2. If, however, $A_2$ moves to room $r$ in region $C$, its local view becomes $(c_{2,r}^1, a_{1,r}^2, b_{2,j}^2)$. Since $b_{2,j} \not\equiv_{c, r} b_{2,r}$, $A_2$ needs to communicate its local state $c_{2,r}^1$ to $A_1$, using which $A_1$ updates its local view to $(a_{1,i}, c_{2,r}^1, a_{1,k}^2)$. By (1), $A_1$ can verify that the agents are now co-stable, so no further communication is needed.

Thus, using OnlineCoAgent-ComReduce, to maintain constraint $P_c$, each agent needs to inform the other only when it moves from one region to a different region. In contrast, using OnlineCoAgent-ComFull (Fig.1) to maintain constraint $P_c$, the agents would have to immediately inform the other each time they move to a different room, and not till they move to a different region. Therefore, if, for example, the agents spend 90% of their event execution time exploring their current region, then compared to using OnlineCoAgent-ComFull, using OnlineCoAgent-ComReduce could save about 90% of their communication bandwidth.

---

**VII. Experimental Evaluation**

We now present an experimental investigation of the effectiveness in bandwidth reduction of OnlineCoAgent-ComReduce. We compare the number of local-state messages communicated between the agents when using OnlineCoAgent-ComReduce (Fig.2) with that when using the benchmark strategy OnlineCoAgent-ComFull (Fig.1). For the experiments, we created different pairs of agent models and different predicate constraints as follows.

**Agent Model:** We randomly created three different pairs of coordinating agent models with 30, 35 and 40 states, and 15, 10, and 20 events, respectively. Each event was randomly specified as either controllable or uncontrollable.

**Inter-agent Constraint:** For each pair of agent models, we randomly created different constraints (CS’s) with varying degrees of permissiveness or restrictiveness imposed on the coordinating agents. Constraint permissiveness is defined by the ratio $\alpha$ of the number of states satisfying the CS to the total number of system states. This constraint permissiveness ratio $\alpha$ approaches 1 when the CS is the most permissive, i.e., most of the system states satisfy the constraint; and approaches 0 when the CS is the most restrictive, i.e., only a few of the system states satisfy the constraint. For our experiments, four test CS’s with $\alpha$ over the representative range of 0.2, 0.4, 0.6 and 0.9 were created.

For each pair of agent models and an inter-agent constraint, we ran the experiment 50 times, each time of 1000 run-time steps, with each run-time step corresponding to an event execution. At each step, we chose a random event from the set of enabled events for execution. After each experiment, we recorded the number of state messages exchanged between the agents when (i) using OnlineCoAgent-ComFull and (ii) using OnlineCoAgent-ComReduce. We then calculated and summarized in Table I the average and standard deviation of the bandwidth reduction (in percent) of OnlineCoAgent-ComReduce over OnlineCoAgent-ComFull.

**TABLE I:** Bandwidth reduction (in %) of OnlineCoAgent-ComReduce over OnlineCoAgent-ComFull for varying degrees of CS permissiveness $\alpha$

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Pair 1</th>
<th>Pair 2</th>
<th>Pair 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>$CS1 (\alpha = 0.2)$</td>
<td>15.92</td>
<td>0.81</td>
<td>12.63</td>
</tr>
<tr>
<td>$CS2 (\alpha = 0.4)$</td>
<td>5.75</td>
<td>0.23</td>
<td>26.31</td>
</tr>
<tr>
<td>$CS3 (\alpha = 0.6)$</td>
<td>28.56</td>
<td>1.47</td>
<td>5.03</td>
</tr>
<tr>
<td>$CS4 (\alpha = 0.9)$</td>
<td>95.79</td>
<td>2.68</td>
<td>83.72</td>
</tr>
</tbody>
</table>

From these simulation results, we make the following observations. Firstly, the average bandwidth reduction ranges from 5.75% to 95.79% with relatively small standard deviations, indicating the effectiveness of the reduced communication strategy. Secondly, it is extremely high for CS4. Since CS4 is the constraint randomly generated with $\alpha = 0.9$, most of the system states satisfy the constraint. Thus the agents are loosely coupled by the constraint. As a result, they were often co-stable during coordination, and seldom needed to communicate with each other. The reduced communication strategy could apparently exploit such loose coupling between the agents and offer a tremendous advantage in terms of bandwidth savings over the full communication strategy.
VIII. RELATED WORK

Among related work under the same discrete-event paradigm, we have earlier discussed the motivation of the on-line approach proposed in this paper, in contrast and in complement to the off-line approach of [6], [7], [8]. We shall now discuss our on-line discrete-event coordination framework in relation to some other formal frameworks for coordinating agents.

As discussed in Section III, a predicate constraint $P_c$ specifies an inter-agent constraint of the fundamental safety type, asserting that no bad states can ever be visited during multiagent interaction. In a different and important development, Yokoo et al. [2], [9] formulate a distributed constraint satisfaction problem (DCSP) for inter-agent constraints of the domain-value type. In their framework, each agent is represented as a variable with an associated domain of values, and an inter-agent constraint to satisfy is equivalent to a set of no goods to exit from, modeling the constraint-violated (or inconsistent) value combinations for the agent variables. An agent state in our framework is reminiscent of an agent domain value in DCSP. However, the essence of coordination in our work is to completely avoid entering the set of bad multiagent (or composite) states induced by a predicate constraint $P_c$, whereas the essence of a DCSP mechanism is to exit from the set of no goods induced by a domain-valued constraint. In DCSP, an agent action is a domain value whereas in our framework, it is enabling or disabling an event. In what follows, the focus in DCSP is to enable multiple variable agents to cooperatively search for a combination of their actions that satisfies a given domain-valued constraint. Our focus is to enable discrete-event agents to cooperatively compute their coordinating actions so as to always satisfy the supremal coordinable predicate $P_c^{sup}$ of a given safety constraint $P_c$. Satisfying $P_c^{sup}$ - which corresponds to staying within the largest feasible good state subset of the agents' composite state space - allows these agents to have maximal autonomy over their own actions during interaction, as explained in Section III. In DCSP, agent autonomy could manifest itself in the form of weak commitment to action selection [2]. All in all, in parallel with DCSP [2], [9], our coordination framework provides a new distributed constraint satisfaction foundation for multiagent cooperation research.

In another direction, researchers on formal models of multiagent systems increasingly focus their attention on extensions of a formal agent model called Markov Decision Process (MDP) to multiagent settings [10], [11], [12]. However, unlike most of these research efforts, our work considers state transitions as explicit events in the system transitional structure. That enables interesting characteristics of agents to be modeled using the properties of events. For instance, the autonomy of coordinating agents can be modeled using controllable and uncontrollable events, as explained in Section III. Generally, event-based agent models are applicable to a wide range of service systems, including manufacturing, communication and logistics systems [4].

The problem of communication reduction has also attracted increasing attention in recent years. For example, Shen and Lesser [13] and Seow et al. [6], [8] develop algorithms to construct near optimal agent communication strategies for their coordination problems, but their algorithms require off-line planning which may be expensive. Dutta et al. [14] develop a selective communication strategy which, however, cannot guarantee coordination quality. In contrast, our contributions include novel on-line coordination strategies that guarantee coordination quality in some specific sense defined, including one that can achieve significant savings in communication bandwidth, as theoretically proved in Theorem 2 and Theorem 3 and empirically verified in Section VII.

IX. CONCLUSION

This paper has presented new coordination results formalizing how discrete-event agents can interact and communicate in an on-line fashion to guarantee the invariance of a predicate specifying an inter-agent constraint. Specifically, the necessary and sufficient condition of predicate coordinability for two coordinating agents to meet a given predicate is established (Theorem 1); an optimal policy by which the agents can coordinate to maintain the supremal coordinable predicate of a given predicate is presented (Theorem 2); and on-line coordination strategies with full and reduced communication to implement the optimal policy are proposed. Finally, as demonstrated by experimental evaluation, compared to the former, the latter strategy can achieve significant bandwidth reduction while still guaranteeing coordination quality (Theorem 3).

REFERENCES