Coordination Planning: Applying Control Synthesis Methods for a Class of Distributed Agents

Kiam Tian Seow, Manh Tung Pham, Chuan Ma, and Makoto Yokoo

***Abstract*— This paper proposes a new multiagent planning approach to logical coordination synthesis that views a class of distributed agents as discrete-event processes. The coordination synthesis problem involves finding a coordination module for every agent, using which their coordinated interactions would never violate some specified inter-agent constraint. The paper first shows explicitly that, though conceptually different, the well-researched problem of supervision in control science and the problem of distributed agent coordination planning in computer agents science are mathematically related. This basic result enables the application of the vast body of knowledge and associated synthesis tools already founded in discrete-event control theory for automatic coordination synthesis of distributed agents. Within this logical framework, a basic planning methodology applying the discrete-event control synthesis methods is proposed, and illustrated using TCT, a software design tool implementing these methods. A simple example demonstrates how it supports formal synthesis of coordination modules for distributed agents. Discussions in relation to previous work examine the relative significance of the new multiagent planning framework.**

***Index Terms*— Multiagent Planning, Coordination Design, Control Synthesis, Discrete-Event Systems, Automata**

I. Introduction

Rapid advances in information and communication technologies are providing a new infrastructural and communications basis that opens up new challenges to developing complex automated systems more effectively. These systems could offer a richer variety of new or improved services in transportation, telecommunication, education, finance, electronic commerce, manufacturing and defence. The field of *Multiagent Systems* defines a research framework to tackle these challenges by viewing a system as an environment of distributed interacting agents - entities capable of flexible autonomous actions [1, Ch. 1].

In developing effective multiagent systems, one of the key challenges is how to synthesize agents that could *coordinate* their activity, i.e., manage among themselves the interdependent constraints [2] that must exist among them. In essence, this involves deciding which agent does what over time when sharing limited resources, so that they can accomplish their assigned tasks in complex environments. This multiagent coordination problem is important because of the increasing need to deploy computer agents to operate autonomously in distributed electronic environments, where the agents often need to coordinate on a variety of tasks. But in order to build coordination into such systems that could eventually benefit from agent-inspired features such as higher flexibility, autonomy, scalability, reliability and fault tolerance, the use of more formal approaches to laying a strong foundation for system planning and design is necessary.

This paper introduces a formal, domain independent framework for the problem of distributed agent (coordination) synthesis. It can be viewed as a multiagent planning approach [3], [4] to coordination as follows: The starting point is that we are given a system of free agents that need to coordinate, subject to a set of inter-agent constraints. The free (or unconstrained) but fixed-by-design behavior of each agent is prescribed by an automaton [5] interpreted as a discrete-event process (DEP) [6]. From the agent planning viewpoint, this behavior is viewed as an individual agent's fixed local plan, formulated independently to encompass all possible (but not necessarily desirable) local ways to achieve its own goals (or complete its own design tasks). Each inter-agent constraint is also modeled by an automaton in terms of the events of the agents. The system of free agents is called a *discrete-event system* (DES) (Sections II-A and II-B). The fundamental multiagent planning problem then is to synthesize a coordination module for each agent, using which the coordinated interactions among them would never violate some specified inter-agent constraint (Section III). As it turns out, this problem is equivalent to the problem of synthesizing a supervisory controller (Section II-C) for a system of interacting DEPs [6]. A major implication of significant interest is that we can now adapt and apply the vast body of knowledge and associated synthesis tools from *supervisory control* [7], [8] for the automatic synthesis of coordinating agents (Section IV). This will be illustrated by an example (Section V), with a discussion distinguishing the coordination framework from related research (Section VI) and a conclusion summarizing the paper along with some future work (Section VII).

II. Background on Supervisory Control of DES

In this section, we review the essential concepts and supervisory control synthesis methods, along with the relevant procedures of a software design package called TCT [9] that implements these methods [6], [7].

A. Languages and Automata

Let Σ be a finite alphabet of symbols that we refer to as events. A *string* (word) is a finite sequence of events from Σ. Denote Σ^* as the set of all finite strings of elements of Σ. Let ϵ denote the empty string (sequence with no events). Trivially, $\epsilon \in \Sigma^*$. A string s' is a *prefix* of s if $(\exists t \in \Sigma^*)s't = s$.

A *formal language* L over Σ is any subset of Σ^*. Say L_1 is a *sublanguage* of L_2 if $L_1 \subseteq L_2$. The *prefix closure* $\bar{L}$ of L is the language consisting of all prefixes of strings of L. Clearly $L \subseteq \bar{L}$, because any string s in Σ^* is a prefix of itself. A language L is *closed* if $L = \bar{L}$.

K.T. Seow and M.T Pham are with the Division of Computing Systems, School of Computer Engineering, Nanyang Technological University, Republic of Singapore 639798. {asktseow,PHAM0028}@ntu.edu.sg

C. Ma is with the Systems Control Group, The University of Toronto, Toronto, ON M5S 1A4, Canada. cma@control.utoronto.ca

M. Yokoo is with the Department of Intelligent Systems, Faculty of Information Science and Electrical Engineering, Kyushu University, 744 Motooka, Nishi-ku, Fukuoka 819-0395, Japan. yokoo@is.kyushu-u.ac.jp

Events $\sigma \in NULL$ in strings $s \in \Sigma^*$ can be masked out or erased by *projection* $P_0 : \Sigma^* \to (\Sigma - NULL)^*$ according to

$$P_0(\epsilon) = \epsilon$$
$$(\forall s \in \Sigma^*)(\forall \sigma \in \Sigma)$$
$$P_0(s\sigma) = \begin{cases} P_0(s)\sigma, & \text{if } \sigma \in \Sigma - NULL \\ P_0(s), & \text{otherwise.} \end{cases}$$

The action of P_0 on a string $s \in \Sigma^*$ is to erase all the occurrences of $\sigma \in NULL$ in string s. P_0 is the natural projection of Σ^* onto $(\Sigma - NULL)^*$.

A language usually has infinite number of strings. However, if the language is *regular* [5], then it can be *generated* by an automaton with a finite state set. An *automaton* A is a 5-tuple $(Q, \Sigma, \delta, q_0, Q_m)$ where (i) Q is the finite set of states, (ii) Σ is the finite set of events, (iii) $\delta : \Sigma \times Q \to Q$ is the (partial, deterministic) transition function, (iv) q_0 is the *initial state*, and (v) $Q_m \subseteq Q$ is the subset of *marker states*.

An automaton can be represented graphically by an edge-labelled directed graph with states represented by nodes, a transition $\delta(\sigma, p) = q$ by an edge going from the state p to q with an event labelled by σ, the initial state by an entering arrow, and the marker states by darkened nodes. We write $\delta(\sigma, p)!$ to denote that a transition $\delta(\sigma, p)$ is defined.

The definition of δ can be extended to Σ^* as follows: $\delta(\varepsilon, q) = q$ and $(\forall \sigma \in \Sigma)(\forall s \in \Sigma^*)\delta(s\sigma, q) = \delta(\sigma, \delta(s, q))$.

The behavior may then be described by two languages: $L(A) = \{s \in \Sigma^* : \delta(s, q_0)!\}$ and $L_m(A) = \{s \in L(A) : \delta(s, q_0) \in Q_m\}$. $L(A)$ is called the prefix-closed language generated by automaton A, and $L_m(A)$, the language marked by automaton A.

By definition, $L_m(A) \subseteq L(A)$, i.e., it is a sublanguage of $L(A)$ whose strings end in a state of Q_m, and is a distinguished subset. If automaton A represents a DES, then Q_m is meant to represent completed 'tasks' and $L_m(A)$, sequences of tasks carried out by the physical process that the model A is intended to model [6]. If automaton A models a constraint (behavioral) specification K, then $K = L_m(A)$ is the *behavior of interest*.

A state $q \in Q$ is *reachable* (from the initial state q_0) if there exists a string $s \in \Sigma^*$ such that $\delta(s, q_0) = q$. Similarly, a state $q \in Q$ is *coreachable* if there exists a string $s \in \Sigma^*$ such that $\delta(s, q) \in Q_m$. Then automaton A is *trim* if every state in Q is both reachable and coreachable. A is said to be *nonblocking* if every reachable state in Q is coreachable, i.e., $\overline{L_m(A)} = L(A)$. Otherwise, A is *blocking*. In particular, A is nonblocking if it is *trim*. If A is not trim, the procedure $\mathsf{Trim}(A)$ in TCT returns a trimmed automaton that generates the same marked language as A.

On 'equivalence' of two automata A_1 and A_2, we write $A_1 = A_2$ if their edge-labelled directed graphs are identical in structure (including marker states); and $A_1 \equiv A_2$ if the automata generate the same prefix-closed and marked languages. So $(A_1 = A_2) \overset{(\text{implies})}{\Longrightarrow} (A_1 \equiv A_2)$ but the converse is not true in general. Finally, we write $A_1 \sqsubseteq A_2$ if automaton A_1 is nonblocking and generates a marked sublanguage of A_2.

In TCT, the procedure Create allows the input of automata (in a certain format). The projection P_0 for regular languages (i.e., languages generated by automata) is implemented by the procedure P0.

B. Composition of Automata

1) The Synchronous Product: Consider an automaton G modeling (the behavior of) a DES. A complex DES model G is usually modeled as a system of several interacting discrete-event processes (DEP's), each modeled by an automaton A_i and composed together using the *synchronous* operator $\|$. The system G is thus a synchronous product defined as follows.

Let $A_i = (Q^{A_i}, \Sigma^{A_i}, \delta^i, q_0^i, Q_m^{A_i}), i = 1, 2$, be two automata. Then $G = (Q, \Sigma, \delta, q_0, Q_m)$, as the *synchronous product* [10] of A_1 and A_2 denoted by $G = A_1 \parallel A_2$, is synthesized with (i) $\Sigma = \Sigma^{A_1} \cup \Sigma^{A_2}$, (ii) $Q = Q^{A_1} \times Q^{A_2}$, (iii) $Q_m = Q_m^{A_1} \times Q_m^{A_2}$, (iv) $q_0 = (q_0^1, q_0^2)$, and (v) $\delta = \delta^1 \times \delta^2$ defined by

$$\delta(\sigma, (q^1, q^2)) = \begin{cases} (\delta^1(\sigma, q^1), \delta^2(\sigma, q^2)), & \text{if } \sigma \in \Sigma^{A_1} \cap \Sigma^{A_2} \text{ and} \\ & \delta^1(\sigma, q^1)! \text{ and } \delta(\sigma, q^2)! \\ (\delta^1(\sigma, q^1), q^2), & \text{if } \delta^1(\sigma, q^1)! \text{ and } \sigma \notin \Sigma^{A_2} \\ (q^1, \delta^2(\sigma, q^2)), & \text{if } \delta^2(\sigma, q^2)! \text{ and } \sigma \notin \Sigma^{A_1} \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Intuitively, the synchronous product of A_1 and A_2 models DES G, of A_1 and A_2 operating concurrently by interleaving events generated by A_1 and A_2 with synchronization on shared events $\sigma \in \Sigma^{A_1} \cap \Sigma^{A_2}$, such that

- events common to both the automata can occur only if each automata is in a state where such an event is defined;
- events that are not common to both the automata may occur as long as they occur in a sequential order along which they are defined by the respective transition functions of A_1 and A_2.

In TCT, the operator $\|$ is implemented by the procedure Sync. So $\mathsf{Sync}(A_1, A_2)$ returns the synchronous product $A_1 \parallel A_2$, which in general may be blocking even when A_1 and A_2 are not.

2) Cartesian Product: Let $S \sqcap G$ denote the cartesian operation between two automata S and G, such that $L(S \sqcap G) = L(S) \cap L(G)$ and $L_m(S \sqcap G) = L_m(S) \cap L_m(G)$.

If $\Sigma^S = \Sigma^G$, the synchronous operation $\|$ between S and G reduces to the cartesian operation $\sqcap$ between the two and equivalently, we can write $S \parallel G \equiv S \sqcap G$.

In TCT, the operator $\sqcap$ is implemented by the procedure Meet. So $\mathsf{Meet}(S, G)$ returns the cartesian product $S \sqcap G$ which in general need not be nonblocking.

C. Controllability and Supervision

Let $G = (Q, \Sigma, \delta, q_0, Q_m)$ be a DES (G can be built by the synchronous product of a set of simpler automata), and another automaton C specify the desired constraint over the entire event set Σ. The basic control problem is to modify G such that the modified DES G' is nonblocking and only generates strings belonging to $L(C)$.

The basic control framework partitions the event set Σ into two disjoint subsets: the set Σ_c of *controllable* events that can be disabled (or inhibited) by an external controller and the set

Σ_u of *uncontrollable* events that can never be. In an edge-labelled directed graph, a controllable event may be indicated by an optional tick on an edge ($\circ\!\!-\!\!\!\!\!\!\rightarrow\!\!\circ$) representing it. In the control context, a controller can restrict (and therefore modify) the system behavior by only disabling controllable events. We call such a controller a *supervisor* (or supervisory controller). Formally, a supervisor is a function $V : L(G) \to \Gamma$ with $\Gamma := \{\gamma \in 2^{\Sigma} | \gamma \supseteq \Sigma_u\}$. Only events in $V(s) \in \Gamma$ are enabled (and allowed to occur) following the execution of $s \in L(G)$. Write V/G for 'G under the supervision of V.' The *closed behavior* generated by V/G is defined recursively as follows:

1) $\epsilon \in L(V/G)$,
2) if $s \in L(V/G)$, $\sigma \in V(s)$, and $s\sigma \in L(G)$ then $s\sigma \in L(V/G)$,
3) no other strings belong to $L(V/G)$.

Equivalently, the supervisor V can be more abstractly represented by an automaton $S = (X^S, \Sigma^S, \delta^S, x_0^S, X_m^S)$ with $\Sigma^S = \Sigma$, so that $L(V/G) = L(S \sqcap G)$, and S and G are interconnected in a closed-loop feedback configuration [see Fig. 1(a)]. Corresponding to Γ, S is said to be Σ_u-enabling, i.e., $(\forall s \in \Sigma^*)(\forall \sigma \in \Sigma_u)\ s \in L(S \sqcap G)\ \&\ s\sigma \in L(G) \Longrightarrow s\sigma \in L(S \sqcap G)$. The *marked behavior* of G under the supervision of S is $L_m(S \sqcap G)$. S is said to be *nonblocking* (for DES G) if $\overline{L_m(S \sqcap G)} = L(S \sqcap G)$.

Formally, a general problem statement of supervisory control may be given as follows:

Problem 1: Given a DES G and a control constraint C, find a supervisor S such that $S \sqcap G \sqsubseteq C$.

In addressing the problem, a fundamental theorem states that a nonblocking supervisor S for G exists such that the controlled behavior $L_m(S \sqcap G) = K$, where $K = L_m(C) \cap L_m(G)$, if and only if C is *controllable* [6] with respect to G, namely $\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K}$. (Here the notation $\overline{K}\Sigma_u$ denotes the set of strings of the form $s\sigma$ with $sl \in K$ for some $l \in \Sigma^*$ and $\sigma \in \Sigma_u$.) In other words, C is controllable (with respect to G) provided no $L(G)$-string which is already a prefix of $L(C) \cap L_m(G)$, that when followed by an uncontrollable event in G, would exit from the bounds of $\overline{L_m(C) \cap L_m(G)}$; the prefix closed language $\overline{L_m(C) \cap L_m(G)}$ is invariant under the occurrence of uncontrollable events in G.

Automaton C is one such nonblocking supervisor S that exists if C is controllable and C and G are *nonconflicting*, namely, $L(C) \cap L(G) = \overline{L_m(C) \cap L_m(G)}$, i.e., every string in $L(C \sqcap G)$ can be completed to a string in $L_m(C \sqcap G)$. If C is not controllable, there always exists a controllable automaton[1] S generating the *largest* closed sublanguage of $L(C) \cap L(G)$ and is nonconflicting with G [6]. Thus S is nonblocking, generating (with G) the largest marked sublanguage of $L_m(C) \cap L_m(G)$. It follows that $S \sqcap G \sqsubseteq C$, and S is called a supremal controllable and nonblocking automaton of C (with respect to G).

In TCT, a supervisor synthesis algorithm [11] is implemented by the procedure Supcon. $\mathsf{Supcon}(C, G)$ returns a nonblocking supervisor automaton denoted by S_G, with $S_G \equiv (S \sqcap G)$. The trim automaton S_G can be larger in state size than is necessary to achieve the same coordinating actions because it has 'embedded' in it all the *a priori* transitional constraints embodied in the free behavior of the system G itself, as well as some auxiliary constraints. TCT provides Supreduce, a heuristic reduction procedure of polynomial complexity that can often find a greatly state-reduced supervisor S [12]. The procedure also provides a lower bound on the state size of the minimal state supervisor, and the computed supervisor S is actually minimal state if its size matches this bound.

[1]Note, however, that the language of S may be an empty set $\emptyset$.

D. Minimally Reactive Supervision

In the rest of this paper, unless otherwise stated, an automaton is assumed to be trim. Let $E(C)$ denote the non-empty set of automata, such that $S \in E(C)$ iff (i) $(S \sqcap G) \equiv S_G = \mathsf{Supcon}(C, G)$ and (ii) $\Sigma^S = \Sigma^{(S_G)}$. And let $\mathrm{minQ}E(C)$ denote the subset of automata of minimum state size in $E(C)$. Formally,

$$\mathrm{minQ}E(C) = \{S \in E(C) \mid (\forall S' \in E(C))|X^S| \leq |X^{S'}|\}.$$

Let $\Sigma_{loop}^S \subseteq \Sigma^S$ denote the set of all strictly self-loop events in a supervisor automaton S; such an event would never bring S from one state to a different state, i.e.,

$$(\forall \sigma \in \Sigma_{loop}^S)(\forall x \in X)(\delta^S(\sigma, x)! \Longrightarrow \delta^S(\sigma, x) = x).$$

Following, let $\mathrm{maxL}E(C)$ denote the set of automata with the largest set of strictly self-loop events in $E(C)$. Formally,

$$\mathrm{maxL}E(C) = \{S \in E(C) \mid (\forall S' \in E(C))|\Sigma_{loop}^{S'}| \leq |\Sigma_{loop}^S|\}.$$

Let S^* be the automaton that has the largest number of strictly self loop events among all the automata in $\mathrm{minQ}E(C)$. In other words, $S^* \in \mathrm{minQ}E(C)$ and

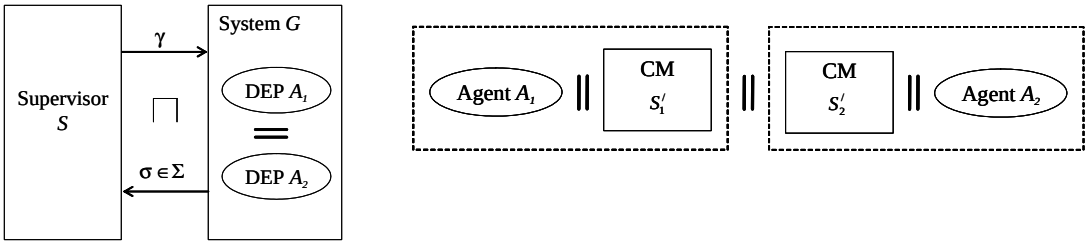
$$(\forall S' \in \mathrm{minQ}E(C))|\Sigma_{loop}^{S'}| \leq |\Sigma_{loop}^{S^*}|.$$

Clearly, $S^* \in \mathrm{minQ}E(C) \cap \mathrm{maxL}E(C)$ provided $\mathrm{minQ}E(C) \cap \mathrm{maxL}E(C) \neq \emptyset$.

Intuitively, a supervisor S^*, with the *least* number of states and the *largest* number of strictly self-loop events, *reacts the least* in the sense that it induces the most memory-state efficient control that need not respond to the largest number of events of DES G to achieve $S_G \sqsubseteq C$. It is said to be *minimally reactive* (under G-equivalence for C). As will be explained later in Section III, such an automaton is found to have an important implication for coordination between two agents.

III. Multiagent Coordination Planning

The basic multiagent coordination planning problem considered in this paper is to modify a system of n interacting agents such that the modified system as a whole is nonblocking and conforms to some inter-agent constraint. The free behavior of each agent A_i is modeled as an automaton interpreted as a DEP. The inter-agent constraint is qualitative (non-numerical) and also specified by some automaton C over the events of all the agents. The basic coordination problem is to modify the multiagent system $G = \|_{i=1}^n A_i$ such that the modified system G' is nonblocking and only generates strings belonging to $L(C)$.



(a) Supervision of two DEP's A_1 and A_2 (b) Coordination of two agents A_1 and A_2

Fig. 1. Illustrating the basic difference: The notion of event-feedback fundamental to implementing supervisory control is absent in multiagent coordination. Following, for the supervisory control problem, the supervisor S is the required solution but for the multiagent coordination problem, the coordination modules S_1' and S_2' are the required solution.

Note that the coordination and supervisory control problems are *conceptually* different. The latter achieves conformance to constraints through an external supervisor interacting with the system of DEP's by event enablement or disablement based on information *feedback* on the occurrence of events [see Fig. 1(a)]. But the former does so by the agents interacting among themselves through their coordination modules (CM's) [see Fig. 1(b)]. Unlike event-feedback control, coordinating interaction also involves synchronous event message passing, where each agent sends messages (for events executed under its jurisdiction) to, and receives relevant event messages from the other agents. This form of communication is necessary to maintain conformance to the specified inter-agent constraint.

Consider two automata A_i and S_i', with $\Sigma^{A_i} \subseteq \Sigma^{S_i'}$. The event set of A_i are classified into controllable event set $\Sigma_c^{A_i}$ and uncontrollable event set $\Sigma_u^{A_i}$. Then S_i' can be a CM of agent A_i provided S_i' is $\Sigma_u^{A_i}$-enabling, i.e.,
$(\forall s \in (\Sigma^{S_i'})^*)(\forall \sigma \in \Sigma_u^{A_i})\ s \in L(A_i \parallel S_i')\ \&\ P_0(s)\sigma \in L(A_i) \implies s\sigma \in L(A_i \parallel S_i')$, where P_0 is the natural projection from $(\Sigma^{S_i'})^*$ to $(\Sigma^{A_i})^*$.

Informally, $\Sigma_u^{A_i}$-enabling means that when agent A_i is coordinating (via $\parallel$) through S_i', its uncontrollable events can never be prevented from occurring.

Formally, a general problem statement of multiagent coordination planning may now be given as follows:

Problem 2: Given a system $G = \|_{i=1}^n A_i$ of n agents and an inter-agent constraint C, find a CM S_i' for each agent A_i such that $\|_{i=1}^n (A_i \parallel S_i') \sqsubseteq C$.

Interestingly, it turns out that the Control Problem 1 and Coordination Problem 2 are *mathematically* equivalent in the sense of Theorem 1 below. The theorem presents the result for two agents, but can be easily extended to multiple agents. As we will show and explain, this theoretical insight and existing control synthesis methods help us to focus on the synthesis of agent CM's - the modification solution - that have the following nice properties:

1) The agent CM's are minimally interventive, meaning that the coordinating agents do not disable their own controllable events unless not doing so will lead to violation of the inter-agent constraint C.
2) The number of events to be communicated via CM's between the agents is relatively small. This property is necessary when the underlying communication infrastructure has limited capability or the communication cost is high.
3) Each agent CM can be efficiently implemented in terms of a relatively small number of memory states.

A class of such agent CM's, called *minimal* coordination modules, will be formally described later.

Theorem 1: Given automata S, A_1 and A_2, with $\Sigma^S = \Sigma^{A_1} \cup \Sigma^{A_2}$,

$$S \sqcap (A_1 \parallel A_2) \equiv (A_1 \parallel S_1') \parallel (S_2' \parallel A_2)$$

and

$$\Sigma^{S_1'} \cap \Sigma^{S_2'} = (\Sigma^S - \Sigma_{loop}^S) \cup (\Sigma^{A_1} \cap \Sigma^{A_2}),$$

where S_i', $i \in \{1, 2\}$, is automaton S, but with all its strictly self-loop events not defined in agent A_i, i.e., $\sigma \in \Sigma_{loop}^S - \Sigma^{A_i}$, removed.

Proof: Since S and $A_1 \parallel A_2$ share the same event set Σ^S and $\parallel$ is associative and commutative, it follows that

$$\begin{aligned} S \sqcap (A_1 \parallel A_2) &\equiv S \parallel (A_1 \parallel A_2) \\ &\equiv S \parallel A_1 \parallel A_2 \\ &\equiv A_1 \parallel S \parallel A_2. \end{aligned}$$

And since S_i' is S but with all its strictly self-loop events in S not defined in agent A_i, i.e., $\sigma \in \Sigma_{loop}^S - \Sigma^{A_i}$, removed, it follows by the definition of the *synchronous* operator $\parallel$ (see Section II-B.1) that $S \equiv S_1' \parallel S_2'$. Thus,

$$\begin{aligned} S \sqcap (A_1 \parallel A_2) &\equiv A_1 \parallel (S_1' \parallel S_2') \parallel A_2 \\ &\equiv (A_1 \parallel S_1') \parallel (S_2' \parallel A_2). \end{aligned}$$

By set-theoretic manipulations,

$$\begin{aligned} \Sigma^{S_1'} \cap \Sigma^{S_2'} &= \big\{\Sigma^S - (\Sigma_{loop}^S - \Sigma^{A_1})\big\} \cap \big\{\Sigma^S - (\Sigma_{loop}^S - \Sigma^{A_2})\big\} \\ &= \big\{(\Sigma^S - \Sigma_{loop}^S) \cup (\Sigma_{loop}^S \cap \Sigma^{A_1})\big\} \\ &\quad \cap \big\{(\Sigma^S - \Sigma_{loop}^S) \cup (\Sigma_{loop}^S \cap \Sigma^{A_2})\big\} \\ &= (\Sigma^S - \Sigma_{loop}^S) \cup (\Sigma^{A_1} \cap \Sigma^{A_2} \cap \Sigma_{loop}^S) \\ &= \big\{(\Sigma^S \cup \Sigma^{A_1} \cap \Sigma^{A_2}) - \Sigma_{loop}^S\big\} \\ &\quad \cup (\Sigma^{A_1} \cap \Sigma^{A_2} \cap \Sigma_{loop}^S) \\ &= (\Sigma^S - \Sigma_{loop}^S) \\ &\quad \cup \big\{(\Sigma^{A_1} \cap \Sigma^{A_2} - \Sigma_{loop}^S) \cup (\Sigma^{A_1} \cap \Sigma^{A_2} \cap \Sigma_{loop}^S)\big\} \\ &= (\Sigma^S - \Sigma_{loop}^S) \cup (\Sigma^{A_1} \cap \Sigma^{A_2}). \end{aligned}$$

Hence the result. ■

Theorem 1 may be interpreted as follows: Suppose automaton $G = A_1 \parallel A_2$ models a system. Then the left-hand side (of Theorem 1) can be viewed as a supervisor S controlling the system G (of possibly interacting DEP's A_1 and A_2) if S is controllable with respect to G. Supervisor S is *nonblocking* (with respect to G) if $S_G \equiv S \sqcap G$ is *trim*. The right-hand side can be viewed as distributed agents, each (with its free behavior) modeled by A_i, $i \in \{1, 2\}$, coordinating between themselves via their respective coordination modules S_i' that each is deemed to be 'equipped' with.

By definition, Σ_{loop}^S is the set of all strictly self-loop events for a given S, and these events are said to be 'irrelevant' to S as a supremal controllable and nonblocking automaton of some inter-agent constraint C, but which can occur in the agents composed as G. Being inherently synchronous in the free interactions between the agents A_1 and A_2, every shared event $\sigma \in \Sigma^{A_1} \cap \Sigma^{A_2}$ must be included for synchronization, even if it is irrelevant to achieving conformance to constraint S, i.e., $\sigma \in \Sigma_{loop}^S$. It follows that $[\Sigma_{loop}^S - (\Sigma^{A_1} \cap \Sigma^{A_2})]$ is the largest event set that *need not* be communicated between the agents to achieve conformance to the (controllable) inter-agent constraint S. However, internal communication with the local events of each agent A_i is needed. Therefore, as formally established, $\Sigma_{loop}^S - \Sigma^{A_i}$ is the largest set of events that can be removed from a given S to yield S_i', and yet $(A_1 \parallel S_1') \parallel (A_2 \parallel S_2')$ preserves $S \parallel G$. Importantly, being equipped with S_i' means that the agents A_i's need to synchronize with each other only on the set of events Σ_{sync}^S, $\Sigma_{sync}^S = \Sigma^{S_1'} \cap \Sigma^{S_2'}$ (all events except the set of unshared and strictly self-loop events in S).

In the context of coordination, the controllable $(S \parallel G) \sqsubseteq C$ is the correct and complete (feasible) coordination subspace of C in the presence of uncontrollable events in the multiagent system G; and $(S \parallel G)$ and $(S \sqcap G)$ (the control space) are equivalent since $\Sigma^S = \Sigma$. This suggests that we can apply existing control synthesis methods to obtain a supremal controllable S (which is nonblocking), from which every S_i' can be obtained as implied in Theorem 1. It means we can obtain coordination modules for agents in a system G such that the coordinating behavior $S \parallel G$ is nonblocking and does not contradict the inter-agent constraint C. As $S_1' \parallel S_2' = S$, the coordination is minimally interventive, meaning that the coordinating agents will not unnecessarily disable their own controllable events since the admissible coordination space $S \parallel G$ (due to the supremal controllable and nonblocking automaton S of C [8]) is feasibly the least constrained that still conforms to the specified inter-agent constraint C.

The size and events of Σ_{loop}^S may change, and so may those of Σ_{sync}^S for an S of a different state size in $E(C)$. If $S \in \mathrm{minQ}E(C) \cap \mathrm{maxL}E(C)$ (which means it is of minimum state size and has the largest Σ_{loop}^S) and $\Sigma_{loop}^S \bigcap (\Sigma^{A_1} \cap \Sigma^{A_2})$ is the smallest [among all automata of $E(C)$], then each module S_i' obtained from such an S is said to be a *minimal* coordination module[2] because of the following additional properties:

1) The common set of events Σ_{sync}^S they collectively offer for synchronization between the two agents would be the *smallest*, inducing efficient communication (and hence a relatively small amount of information to be shared[3]) between the agents when coordinating to satisfy the inter-agent constraint C.
2) The CM S_i' that each agent A_i is equipped with is memory-state efficient, since it is synthesized from a minimum state S.

Unfortunately, finding a supervisor S of minimum state size is already NP-hard [12].

Corollary 1: Given automata S, A_1 and A_2, with $\Sigma^S = \Sigma^{A_1} \cup \Sigma^{A_2}$,

$$S \sqcap (A_1 \parallel A_2) \equiv (A_1' \sqcap S_1') \parallel (S_2' \sqcap A_2')$$

and

$$\Sigma^{S_1'} \cap \Sigma^{S_2'} = (\Sigma^S - \Sigma_{loop}^S) \cup (\Sigma^{A_1} \cap \Sigma^{A_2}),$$

where, for $i \in \{1, 2\}$,

1) A_i' is agent A_i, but with all the events of S_i' not defined in agent A_i, i.e., $\sigma \in \Sigma^{S_i'} - \Sigma^{A_i}$, self-loop adjoined at each state $q \in Q^{A_i}$, and
2) S_i' is automaton S, but with all its strictly self-loop events not defined in agent A_i, i.e., $\sigma \in \Sigma_{loop}^S - \Sigma^{A_i}$, removed.

Proof: For $i \in \{1, 2\}$, since $\Sigma^{S_i'} = (\Sigma^{A_1} \cup \Sigma^{A_2}) - (\Sigma_{loop}^S - \Sigma^{A_i})$, it follows that $\Sigma^{A_i} \subseteq \Sigma^{S_i'}$. This implies $A_i \parallel S_i' \equiv A_i' \sqcap S_i'$, with A_i' as defined. Hence, rewriting Theorem 1, the corollary follows. ■

Corollary 1 offers an intellectually interesting view: A feasible control solution S can be decomposed into a coordination solution with CM's S_1' and S_2', connected in a close interplay of *local control* ($\sqcap$) and *global coordination* ($\parallel$). Control is local in that an individual agent A_i is augmented to only enable or disable events in its *local* set Σ^{A_i}. Coordination is global in that it involves synchronous communication between the agents on events in the *system* subset $\Sigma^{S_1'} \cap \Sigma^{S_2'}$. This interplay can meet some (global) inter-agent constraint C (expressed over the whole event set Σ^S) if S is a supremal controllable and nonblocking automaton of C.

IV. Coordination Planning as Control Synthesis

An important implication of Theorem 1 presented in the preceding section and the discussion that followed is that control synthesis can be adapted and applied as a new multiagent planning approach to coordination. And the approach can be carried out without 'reinventing the wheel' through a planning methodology that we will present in Section IV-C. This approach allows automatic synthesis of minimally interventive coordination modules. These modules guarantee that the respective agents, individually equipped with each, will exhibit a coordinated behavior that is not only correct but feasibly complete with respect to specified inter-agent constraints.

[2]The conditions for minimal coordination specified herein revise and clarify that in the preliminary conference version [13].

[3]The amount of (event) information that the agents may share also depends on how stringent the inter-agent constraint is.

A. Uncontrollable and Controllable Events in an Agent Model

An agent is said to own an event provided its free behavioral model contains the event. Then interpreted slightly differently from the context of control design [see Fig. 1(a)], in the new context of coordinating agent design [see Fig. 1(b)], an agent is said to only enable or disable the events it owns. An event can either be controllable or uncontrollable.

One can think of an uncontrollable event as one that is predetermined to be inherently *autonomous*, i.e., it can occur solely at the *free will* of its owner agent. Such a free will is usually exerted by the engineering dynamics inside the agent's local state where the event is defined. Following the discrete-event modeling of a multiagent system, a system modeler has to pre-specify each event as either controllable or uncontrollable. As a rule, an event is pre-specified as uncontrollable if it is critical to the owner agent such that disabling the event and limiting its autonomy just to conform to an inter-agent constraint is undesirable, expensive or impossible. Machine *breakdown* is a hard example of an uncontrollable event. It can occur at the free will of the machine agent. Here, of course, the free will is exerted by the possibly unrestrained ageing dynamics of the machine agent. Customer *arrival* at a banking ATM is a softer example of an uncontrollable event. It is inhibitable by its owner agent (the customer). But in coordination, not only is always prohibiting such an event unnecessarily restrictive to the agent, it is often also an undesirable decision, hence pre-specified as uncontrollable.

An agent either inhibits or executes its controllable events when interacting with other agents. While the agent can certainly prevent a controllable event from occurring by disabling it, whether or not an enabled event can be executed depends on the agent's underlying mechanism[4]. A system modeler can freely pre-specify an event to be controllable as he sees fit, provided that some mechanism can be implemented for the agent to actually prohibit or execute the event as necessary. An example of an event that may be pre-specified as controllable is a traffic green light *turned on*.

B. Multiagent Coordination and Communication

Under the foregoing notation, in a multiagent setting,

$$\Sigma^S_{sync} = \bigcap_{all\ i} \Sigma^{S'_i} = [(\Sigma^S - \Sigma^S_{loop}) \cup \bigcap_{all\ i} \Sigma^{A_i}].$$

Agent A_i would need to communicate the (occurrence of its local) events in $\Sigma^{A_i} \cap \Sigma^S_{sync}$ to every other agent (via synchronous event message passing), but need not communicate those in $\Sigma^{A_i} \cap (\Sigma^S_{loop} - \Sigma^{A_j})$ to agent A_j, $j \neq i$, nor any other agent it does not inherently share these events with. It however needs to communicate the events in $\Sigma^{A_i} \cap (\Sigma^{A_j} - \Sigma^S_{sync})$ specifically to agent A_j. $\Sigma^{A_i} \cap (\Sigma^{A_j} - \Sigma^S_{sync}) \subseteq \Sigma^S_{loop}$, but is also the subset of events shared inherently between itself and agent A_j and so must be communicated, regardless of the fact that these events are irrelevant to achieving conformance to S.

[4]As Section IV-D will elaborate more on, we can view this as a commitment mechanism.

So in general, every S'_i obtained from an S is a minimal coordination module if all the following criteria hold:

1) $S \in \mathrm{minQ}E(C) \cap \mathrm{maxL}E(C)$;
2) among all automata of $\mathrm{minQ}E(C) \cap \mathrm{maxL}E(C)$,
   a) $\Sigma^S_{loop} \cap \bigcap_{all\ k} \Sigma^{A_k}$ is the smallest;
   b) $(\forall j)\Sigma^{A_i} \cap (\Sigma^{A_j} - \Sigma^S_{sync})$ is the smallest.

Condition 1 ensures that Σ^S_{loop} is the largest. Together with Criterion 2a, Σ^S_{sync} is the smallest. And together with Criterion 2b, for every pair of agents (A_i, A_j), $j \neq i$, the event set $\Sigma^S_{sync} \cup [\Sigma^{A_i} \cap (\Sigma^{A_j} - \Sigma^S_{sync})]$ that each corresponding pair (S'_i, S'_j) collectively offers for synchronous communication between the two agents would be *the smallest*. It should be clear that

$$\Sigma^S_{sync} \cup [\Sigma^{A_i} \cap (\Sigma^{A_j} - \Sigma^S_{sync})] = \Sigma^S_{sync} \cup [\Sigma^{A_j} \cap (\Sigma^{A_i} - \Sigma^S_{sync})],$$

thus clarifying that the set on the left-hand side is the communication event set between agents A_i and A_j for coordination.

In the special case of only two coordination agents, A_1 and A_2, $\Sigma^{A_1} \cap (\Sigma^{A_2} - \Sigma^S_{sync}) = \emptyset$.

A sufficiency condition for Criterion 2b is the following: $(\forall \sigma \in \Sigma)(\forall j, k; j \neq k)$

$$\left(\sigma \in \Sigma^{A_j} \cap \Sigma^{A_k} \Longrightarrow \sigma \in \bigcap_{all\ i} \Sigma^{A_i} \right).$$

Intuitively, it means that any shared event is a common one among all the agents. When this condition holds, Condition 2b is satisfied, since $\Sigma^{A_i} \cap (\Sigma^{A_j} - \Sigma^S_{sync}) = \emptyset$ and thus is the smallest. A special and not uncommon case of the sufficiency condition is the total absence of shared events among agents, i.e., $(\forall j, k; j \neq k)\Sigma^{A_j} \cap \Sigma^{A_k} = \emptyset$. With this special case, $\Sigma^S_{loop} \cap \bigcap_{all\ k} \Sigma^{A_k} = \emptyset$, thus also satisfying Criterion 2a.

C. Coordination Planning Methodology

Based on the proposed approach, dubbed 'planning as control synthesis,' a simple methodology for the design of coordinating agents follows. The proposed methodology supports the following planning steps, which can be clearly prescribed in terms of procedures provided by TCT [9].

Step 1:. Modeling

- Use Create to input all automaton models of the free agents and their inter-agent constraints.
- Use Meet and Sync, procedures for $\sqcap$ and $\|$ respectively, to combine the automata as needed.

Step 2:. Control Synthesis

- Use Supcon:
  - Input: automaton G, a synchronous product on all agent automata A_i and automaton C, a meeting of all inter-agent constraints C_j, with $\Sigma^C = \bigcup_{all\ i} \Sigma^{A_i}$.
  - Output: automaton S_G, a nonblocking supervisor.
- Use Condat:
  - Input: automata G and S_G.
  - Output: control data DAT (i.e., events disabled at each state) for S_G.

- Use Supreduce:
  - Input: automaton S_G and control data DAT.
  - Output: automaton S (state-reduced supervisor).

Step 3:. Coordination Synthesis

- Use Asynevent:
  - Input: automaton S.
  - Output: event set Σ^S_{loop} (set of strictly self-loop events in S).
- Use P0 for each agent automaton A_i:
  - Input: automaton S and event set $\Sigma^S_{loop} - \Sigma^{A_i}$ as the $NULL$ set.
  - Output: automaton S'_i, a coordination module for agent A_i.

The key procedures in the planning steps above have been defined in Section II, and the rest should be clear from their specified input-output. The reader may refer to Wonham [7] for a more detailed description of the TCT procedures used.

The current version of TCT [9] supports synthesis for a system of up to 2 million states. Synthesis of very large systems is a subject of ongoing research (e.g., [14]), and is beyond the scope of this paper. In any case, the proposed methodology applies DES control synthesis methods, and need not depend on TCT which is one available software implementation of the methods.

One obvious coordination solution[5] is $S'_i = S$. In this case, the agents communicate synchronously all of their (local) events to achieve conformance to inter-agent constraints. However, this solution may entail unnecessary events communicated between the agents. Theorem 1 has clearly revealed the possibility of a better coordination solution with regard to less synchronous communication. It is the basis for extending the control methodology (Steps 1 and 2) with Step 3, as in the coordination planning methodology presented above.

Finally, it is worth pointing out that many man-made systems can, at some level as abstraction, be modeled by automata (interpreted as DEPs). As with any design modeling formalism, abstraction is inevitable. But as long as a system's constraint-related features can be described in automata, an *abstraction* of the resulting system (of agents) can be modeled and the coordination planning methodology can be applied. Within this formalism, modeling is very flexible. We can use a number of automata to formally describe one agent's behavior, or use a number of them as inter-agent constraints describing the interactions among two or more agents. So the modeling of agent behavior within this planning framework is naturally modular and decentralized.

D. Conventions, Commitments and Joint Intentions

Jennings [4], [15] has hypothesized that *commitments* and *conventions* constitute coordination in multiagent systems. The former are pledges to undertake specific actions; the latter are means of monitoring commitments as the system actually evolves. Accordingly, the proposed planning framework, being very general, only supports the design of agent conventions but lends credentials to the *Centrality of Commitments and Conventions Hypothesis* [15] that all coordination mechanisms can ultimately be reduced to commitments and their associated conventions, as explained below:

Let automaton A be (the free behavior of) a distributed agent, equipped with S', a coordination module synthesized under the proposed methodology. Where a state in $A \parallel S'$ has more than one event defined, the agent is deemed to be capable of *selecting* an event to execute via an underlying mechanism. This mechanism, however, is assumed not modeled in supervisory control theory as originally asserted in [16], and hence in the current proposed planning framework that inherits it. The reason is to develop a coordination theory that is valid for a wide range of applications. Therefore, within the framework, no postulation is made on how the mechanism selects an event for execution. The actual selection might be decided offline or online among the agents through their mechanism. This might be separately designed for a peculiar application domain by a system designer. In connecting to the two concepts that co-found the idea of coordination, an event σ local to agent A (i.e., $\sigma \in \Sigma^A$) that is selected for execution in a state of $A \parallel S'$ is deemed as a commitment by the agent, since the selected event would occur unless preempted by another whose occurrence is uncontrollable. The agent's (local) convention is the language space defined by $A \parallel S'$, since it details all the event sequences in which the agent's commitments can traverse feasibly, i.e., without ever exiting the bounds of the inter-agent constraint on which S' is synthesized.

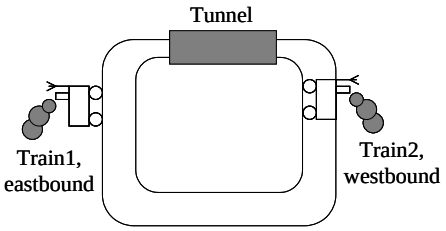
Finally, viewed from a taxonomy of multiagent interactions proposed by Parunak et al [17], our proposed framework can be regarded as collaboration defined as 'coordination based on direct communication plus joint intentions.' In our coordination paradigm, the agents are designed to synchronize or directly communicate with each other on a subset of their events to carry out their *joint intentions*. The inter-agent constraint can be said to specify their joint intentions.

V. Illustrative Example

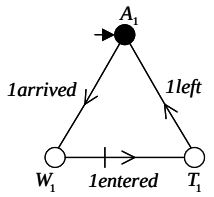
In this section, we present an example to illustrate and discuss the use of the proposed TCT-based planning methodology for coordination design of a multiagent system.

The system under study is a different version of the train controller system adapted in [18]. As shown in Fig. 2(a), the distributed system has two train agents, modeled by automata T_1 and T_2 as shown in Figs. 2(b) and 2(c), but *no central controller*. The modeled train behaviors should be quite self-explanatory, with $\Sigma = \Sigma_c \cup \Sigma_u$, where we arbitrarily fix $\Sigma_c = \{1entered, 2entered\}$ and $\Sigma_u = \{1arrived, 1left, 2arrived, 2left\}$. The train agents, one eastbound (EB) and one westbound (WB), each occupies its own loop track. But at one point, both tracks pass through a tunnel. There is no room to accommodate both trains going past each other in the tunnel. Unlike in [18], there are no traffic lights at both ends of the tunnel. Both trains are equipped with a signaller, using which they can send signals to communicate with each other.

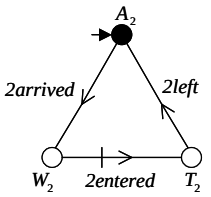
[5]Note that such a trivial solution need not satisfy the conditions for S'_i as in Theorem 1.



(a) Overall structure of train system

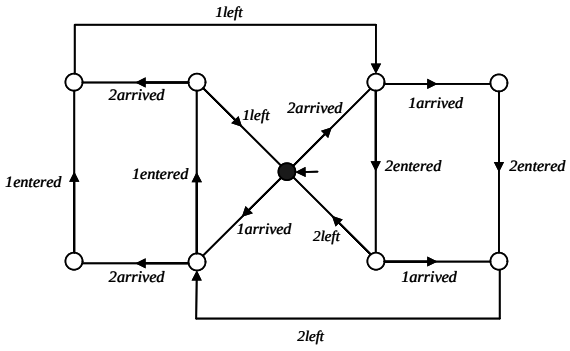


(b) EB train agent T_1



(c) WB train agent T_2

A: **away from-Tunnel**; W: **wait at-Tunnel**; T: **in-Tunnel**



(d) Inter-agent constraint C

Fig. 2. Multiagent coordination planning for a distributed train system

Given this scenario, one possible solution is to let the train agents coordinate between themselves so as to respect the following inter-agent constraint:

> *The train agents are allowed access to the tunnel on a first arrival first access (FAFA) basis, and such that they are never both in the tunnel at the same time.*

The textual description of the desired constraint can be formalized by an automaton C as shown in Fig. 2(d). How this can be done is simply taken for granted here. We refer the reader to the paper [19] on how such an automaton may be more easily determined with the aid of a specification translator. In fact, the reader may convince himself that the automaton C represents the desired constraint. The problem then becomes that of formally synthesizing coordination modules for the train agents. This can be systematically done using the automated planning methodology proposed in Section IV-C.

The result after completing *Step 2* is a simplified supervisor S which, for this system, is found to be minimal state (by Supreduce) and by inspection, can be verified to be an automaton S^* (see Section II-D). It is shown in Fig. 3. Following *Step 3*, we obtain $\Sigma^S_{loop} = \{1entered, 2entered\}$, using which the event $\sigma \in \Sigma^S_{loop} - \Sigma^{T_i}$, $i \in \{1, 2\}$ is projected or masked out from S to obtain the coordination module (CM) S'_i for train agent T_i, as shown in Fig. 4. To elaborate, using these CM's means: agent T_1 must inform agent T_2 whenever any of its events, $1arrived$ and $1left$, occurs, but need not do so when event $1entered$ occurs; and agent T_2 reciprocates in turn. Since $\Sigma^{A_1} \cap \Sigma^{A_2} = \emptyset$, Criterion 2 stated in Section IV-B is satisfied, and so it follows that the resulting CM's are minimal coordination modules.

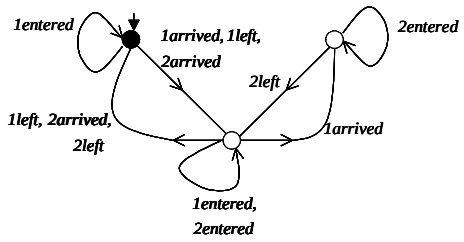


Fig. 3. A simplified (minimally reactive) supervisor S

To meet the specified inter-agent constraint C, a clear advantage of our approach is that it offers unintuitive design insights, informing us the events (namely, $\Sigma^S_{sync} \cap \Sigma^{T_i} = \{iarrived, ileft\}$) that agent T_i, $i \in \{1, 2\}$, would need to communicate to the other agent T_j, $i \neq j \in \{1, 2\}$, via synchronous messaging, and those (namely, $(\Sigma^S_{loop} - \Sigma^{T_j}) = \{ientered\}$) that it need not. In our opinion, without the formal synthesis methodology, even for this simple train example, it would have been quite difficult to determine which events can be in Σ^S_{loop} that need not be coordinated between the train agents.

VI. Related Work

In Agents Research: Wooldridge and Dunne [20], [21] investigate an agent design problem. Basically, the problem investigates the existence (or success) of an abstract agent that can perform some specified achievement and/or maintenance tasks in a given model of a *nondeterministic* environment. Achievement and maintenance tasks are specified as subsets of states in the environment model, and correspond respectively to marker (or goal) states of the system models and (static) constraint specifications in the supervisory control framework [6]. In the Wooldridge and Dunne setting, an agent is similar to a supervisor as in the original supervisory (discrete-event) control setting (see Section II-C). Their existence of an agent for some specified tasks is due solely to *nondeterminism* of the environment (or system) model, which is well understood in the agents literature (e.g., [20]). However, issues of agent existence (and synthesis) due to the presence of uncontrollable events do not naturally arise in their non-control-theoretic setting.

In the Wooldridge and Dunne framework, the focus has been on the computational complexity associated with various task versions of their existence problem. Unlike in the supervisory control framework [7], [9], no constructive method for synthesizing an agent, if it exists, has been proposed to facilitate planning. In the supervisory control framework, agent (or supervisor) synthesis for a nondeterministic environment can be easily handled as well. Two basic approaches are supported:

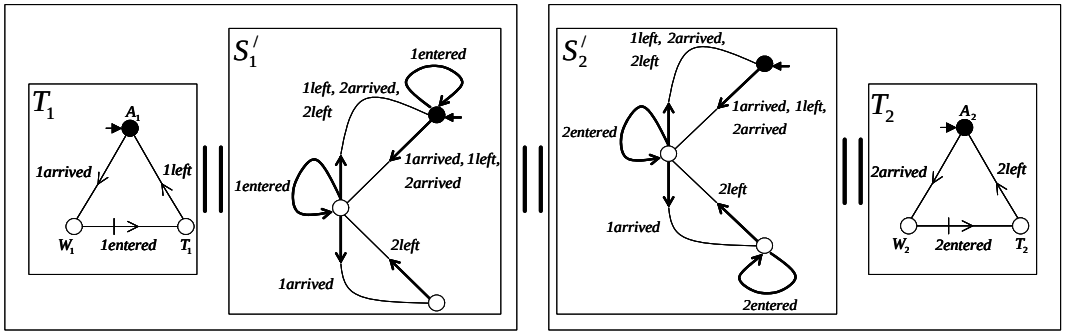


Fig. 4. Coordination modules (CM's) for the distributed train agents

(i) prior to synthesis, convert a nondeterministic model to a deterministic one that is equivalent with respect to language [5], and (ii) perform synthesis directly on a nondeterministic model [14]. The former is a standard conversion approach while the latter is a recent proposal that circumvents the problem of NP-hardness associated with conversion.

Giunchiglia and Traverso [22] and Hoek and Wooldridge [18] use model checking [23] to establish the existence of a group of agents that coordinate to achieve the goals stated in the constraint specifications expressed in branching time temporal logic and alternating temporal epistemic logic, respectively. It essentially involves determining a path in the transition model, associated with the respective approach, of the environment against the specified logic constraints.

All the approaches mentioned above are cast in a setting where all the transitions in their environment models are assumed to be controllable or inhibitable. Existence of an agent [20], [21], [24] or a group of agents [22], [20] has been the key focus in these planning approaches. A *feasible* transition path that exists, achieving the goal stated in the specification, no doubt admits a feasible plan but is often not complete, resulting in it being too restrictive. There are possibly many other (feasible) paths that also achieve the goal, but have been omitted.

Wolper and Manna [25] and Clarke and Emerson [26] synthesize, instead of just a path, a synchronization skeleton of a concurrent system, if it exists, from specifications expressed in linear and branching time temporal logic, respectively. However, their methods also implicitly assume that all the transitions in the skeleton are controllable.

In Control Research: Rudie et al [27] consider a problem where one control agent (or supervisor) communicates with another agent for information so as to *distinguish the states of its automaton for control decision-making or diagnosis*. Since communication may be costly, a strategy to minimize communication between agents is developed. Like theirs, we also seek to minimize communication between agents, but consider a different problem where one coordinating agent communicates with another agent for information so as to *synchronize designated events for meeting a specified inter-agent constraint*.

Rohloff and Lafortune [28] explore issues related to nonblocking verification of similar control agents (with identical structures) that interact through events broadcast over a network. System events are partitioned into global and private events that affect all agents and exactly one agent, respectively. In our work, Σ^S_{sync} corresponds to the global event set and $(\Sigma^S_{loop} - \bigcup_{(\forall j) j \neq i} \Sigma^{T_j})$ corresponds to agent A_i's private event set. However, our global and private event sets are determined based on some global controllable automaton S, whereas theirs are pre-specified independently. Referring to the train example in Section V, based on the supervisor automaton in Fig. 3, $\{1arrived, 1left, 2arrived, 2left\}$ is the global event set and $\{ientered\}$ is agent T_i's private event set. Finally, despite the different setting and formulation, once the control existence conditions are met for, say DES $G = G_1 \parallel G_2$, we get, from [28, p. 2678, Theorem 3] but using our notations, that

$$(S_1' \sqcap G_1) \parallel (S_2' \sqcap G_2) \equiv S_G,$$

where automaton S_G is trim and $K = L_m(S_G) \subseteq L_m(G)$ is the specified global controllable specification; and $\{S_1', S_2'\}$ is called a set of isomorphic module controllers. Corollary 1 can provide complementary insights, namely, with DES $G = G_1 \parallel G_2 = A_1' \parallel A_2' = A_1 \parallel A_2$, how each local control of an individual $\parallel$-component G_i (or A_i') in DES G can be related to global control of the DES in meeting a global controllable specification K.

Finally, Cho and Lim [29] study a new control problem called multiagent supervisory control in the domain of anti-fault propagation in serial production systems. Therein, two adjacent agent supervisors S_{A_i} and $S_{A_{i+1}}$ are synthesized to control their respective discrete-event processes G_i and G_{i+1}, such that the latter supervisor $S_{A_{i+1}}$ along the production line can eliminate any fault propagation from G_i to G_{i+1}.

A Note on Synthesis versus Verification: Design or synthesis is concerned with finding or constructing some *structures* (either agents situated in some environment model or the coordination modules for given agents) that satisfy given constraint specifications. This is related to but different from verification [30], which is concerned with whether or not some *already constructed* structures satisfy the stated specifications such as the nonblocking property [28].

VII. Conclusion

This paper has introduced multiagent coordination planning as control synthesis, motivated by the fundamental connection between control synthesis and coordination planning as manifested by Theorem 1 and depicted in Fig. 1. Importantly, without 'reinventing the wheel,' it points us to a new planning

basis for the formal design of coordinating agents, by using already well-established control synthesis procedures [7], [12]. And for a start, a coordination planning methodology is proposed, expressed in terms of procedures supported by TCT [9]. Using a simple but non-trivial example, we have illustrated the use of the methodology to synthesize coordination modules for distributed agents.

The use of Supreduce [12] is an important step in the coordination planning framework. This heuristic reduction procedure can often find a greatly state-reduced automaton. However, the procedure is in general not guaranteed to output an automaton that is minimally reactive under system equivalence for an inter-agent constraint. Future research work will investigate the issues associated with the existence, structure and synthesis of such automata for constructing minimal coordination modules.

The proposed multiagent coordination framework models the evolution of multiagent systems by interleaving individual agents' events on the synchronous product $\|$. One important generalization is to admit concurrency, in which multiple events can occur simultaneously. Future work could adapt modeling techniques founded on concurrent discrete-event systems [31] and multiagent products [32], [33] to address concurrency issues. This should enable us to design coordinating agents for more sophisticated and realistic applications.

In conclusion, research on formal coordination synthesis is still relatively new. Automata theory provides the general foundation of computer systems, and we have utilized automata - interpreted as DEP's - as the foundation of agents residing in computer systems. We believe that coordination synthesis founded on discrete-event automata, when fully developed through a fruitful interplay of control-theoretic [10] and agent-theoretic [1] ideas, is general and will have wide applicability for a variety of distributed service systems.

REFERENCES

[1] G. Weiss, Ed., *Multiagent System : A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, London, U.K, 1999.

[2] T. W. Malone and K. Crowston, "The interdisciplinary study of coordination," *ACM Computing Surveys*, vol. 26, no. 1, pp. 87–119, March 1994.

[3] E. H. Durfee, V. R. Lesser, and D. D. Corkill, "Trends in cooperative distributed problem solving," *IEEE Transactions on Knowledge and Data Engineering*, vol. 1, no. 1, pp. 63–83, March 1989.

[4] N. R. Jennings, "Coordination techniques for distributed artificial intelligence," in *Foundations of Distributed Artificial Intelligence*, G. M. P. O.'Hare and N. R. Jennings, Eds. John Wiley and Sons, Inc., New York, 1996, pp. 187–210.

[5] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*. Reading, MA : Addison-Wesley, 1979.

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

[7] W. M. Wonham, *Notes on Control of Discrete-Event Systems ECE 1636F/1637S*. Systems Control Group, University of Toronto, Updated 1st July 2004, http://www.control.toronto.edu/cgi-bin/dldes.cgi.

[8] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, January 1989.

[9] W. M. Wonham, *Control Design Software: TCT*. Developed by Systems Control Group, University of Toronto, Updated 1st July 2007, http://www.control.toronto.edu/cgi-bin/dlxptct.cgi.

[10] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, MA, USA, 1999.

[11] W. M. Wonham and P. J. Ramadge, "On the supremal controllable sublanguage of a given language," *SIAM Journal of Control and Optimization*, vol. 25, no. 3, pp. 637–659, May 1987.

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

[13] K. T. Seow, C. Ma, and M. Yokoo, "Multiagent planning as control synthesis," in *Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'04)*, Columbia University, New York City, USA, July 2004, pp. 972–979.

[14] C. Ma and W. M. Wonham, *Nonblocking Supervisory Control of State Tree Structures. Lecture Notes in Control and Information Sciences, Vol 317*. Springer-Verlag, New York, 2005.

[15] N. R. Jennings, "Commitments and conventions : The foundation of coordination in multi-agent systems," *The Knowledge Engineering Review*, vol. 8, no. 3, pp. 223–250, 1993.

[16] W. M. Wonham, "A control theory for discrete-event systems," in *Advanced Computing Concepts and Techniques in Control Engineering. NATO ASI Series, Vol. F47*, M. J. Denham and A. J. Laub, Eds. Springer-Verlag, Berlin, Heidelberg, 1988, pp. 129–169.

[17] H. V. D. Parunak, S. Brueckner, M. Fleischer, and J. Odell, "A preliminary taxonomy of multi-agent interactions," in *Proceedings of The Second International Conference on Autonomous Agents and MultiAgent Systems (AAMAS'03)*, Melbourne, Australia, July 2003, pp. 1090–1091.

[18] W. van der Hoek and M. Wooldridge, "Tractable multiagent planning for epistemic goals," in *Proceedings of The First International Conference on Autonomous Agents and MultiAgent Systems (AAMAS2002)*, Bologna, Italy, July 2002, pp. 1167 – 1174.

[19] K. T. Seow, "Integrating temporal logic as a state-based specification language for discrete-event control design in finite automata," *IEEE Transactions on Automation Science and Engineering*, vol. 4, no. 3, pp. 451–464, July 2007.

[20] M. Wooldridge, "The computational complexity of agent design problems," in *Proceedings of The Fourth International Conference on Multi-Agent Systems (ICMAS2000)*, Boston, MA, U.S.A, July 2000, pp. 341–348.

[21] M. Wooldridge and P. E. Dunne, "Optimistic and disjunctive agent design problems," in *Pre-Proceedings of The Seventh International Workshop on Agent Theories, Architectures and Languages*, Boston, MA, U.S.A, July 2000, pp. 1–14.

[22] F. Giunchiglia and P. Traverso, "Planning as model checking," in *Recent Advances in AI Planning, Lecture Notes in Artificial Intelligence, Vol 1809 - Subseries of LNCS*, L. Cavedon, A. Rao, and W. Wobcke, Eds. Springer-Verlag, Heidelberg, Germany, 1999, pp. 1–20.

[23] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, Cambridge, MA, U.S.A, 2000.

[24] P. E. Dunne, M. Wooldridge, and M. Laurence, "The computational complexity of boolean and stochastic agent design problems," in *Proceedings of The First International Conference on Autonomous Agents and MultiAgent Systems (AAMAS2002)*, Bologna, Italy, July 2002, pp. 976–983.

[25] P. Wolper and Z. Manna, "Synthesis of communicating processes from temporal logic specifications," in *Proceedings of the 1981 Workshop on Logics of Programs*. Springer-Verlag, New York, 1982, pp. 253–281.

[26] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching time temporal logic," in *Proceedings of 1981 Workshop on Logics of Programs*. Springer-Verlag, New York, 1982, pp. 52–71.

[27] K. Rudie, S. Lafortune, and F. Lin, "Minimal communication in a distributed discrete-event system," *IEEE Transactions on Automatic Control*, vol. 48, no. 6, pp. 957–975, June 2003.

[28] K. Rohloff and S. Lafortune, "The control and verification of similar agents operating in a broadcast network environment," in *Proceedings of the 42nd IEEE International Conference on Decision and Control*, Maui, Hawaii, U.S.A, December 2003, pp. 2673 – 2679.

[29] K. H. Cho and J. T. Lim, "Multiagent supervisory control for antifault propagation in serial production systems," *IEEE Transactions on Industrial Electronics*, vol. 48, no. 2, pp. 460–466, April 2001.

[30] M. Wooldridge, "Agent-based software engineering," *IEE Proceedings on Software Engineering*, vol. 144, no. 1, pp. 26–37, 1997.

[31] Y. Li and W. M. Wonham, "Concurrent vector discrete event systems," *IEEE Transactions on Automatic Control*, vol. 40, no. 4, pp. 628–638, April 1995.

[32] P. Hubbard and P. E. Caines, "Initial investigations of hierarchical supervisory control for multi-agent systems," in *Proceedings of the 38th IEEE International Conference on Decision and Control*, Phoenix, Arizona USA, December 1999, pp. 2218–2223.

[33] I. Romanovski and P. E. Caines, “On the supervisory control of multi-agent product systems: Controllability properties,” *Systems and Control Letters*, vol. 56, no. 2, pp. 113–121, April 2007.