Automatica, Vol. 26, No. 5, pp. 833–844, 1990
Printed in Great Britain.

0005–1098/90 $3.00 + 0.00
Pergamon Press plc

A Coordination Theory for Intelligent Machines*

FEI-YUE WANG† and GEORGE N. SARIDIS†‡

A coordination theory for intelligent machines based on a Petri net approach provides an analytical mechanism of control and communication for various intelligent control systems such as intelligent robotic systems and computer integrated manufacturing systems.

Key Words—Intelligent control; intelligent machines; robotics; entropy; coordination; coordination structure; Petri net transducer; Petri net language; synchronous composition.

Abstract—A formal model for the coordination level of intelligent machines is established. The framework of the coordination level investigated consists of one dispatcher and a number of coordinators. The model called *coordination structure* has been used to describe analytically the information structure and information flow for the coordination activities in the coordination level. Specifically, the coordination structure offers a formalism to (1) describe the task translation of the dispatcher and coordinators; (2) represent the individual process within the dispatcher and coordinators; (3) specify the cooperation and connection among the dispatcher and coordinators; (4) perform the process analysis and evaluation; and (5) provide a control and communication mechanism for the real-time monitor or simulation of the coordination process. A simple procedure for the task scheduling in the coordination structure is presented. The task translation is achieved by a stochastic learning algorithm. The learning process is measured with entropy and its convergence is guaranteed. Finally, a case study of the coordination structure with three coordinators and one dispatcher for a simple intelligent manipulator system illustrates the proposed model and the simulation of the task processes performed on the model verifies the soundness of the theory.

1. INTRODUCTION

IN THIS EARLY stage in the development of intelligent machines, methodological issues are both open and central. Different ideas for the formalization of the definitions and the structure of intelligent machines have been proposed by and debated among various researchers (Albus, 1975; Saridis, 1977; Bejczy, 1986; Meystel, 1986; Stephanou, 1986; Pao, 1986; Vamos, 1987; Antsaklis *et al.*, 1988). The approach proposed by Saridis (1977) can be thought of as the result of the intersection of the three major disciplines of *Artificial Intelligence, Operation Research* and *Control Theory*.

The structure of intelligent machines is defined by Saridis (1977, 1983, 1986) to be the structure of hierarchically intelligent control systems, composed of three levels ordered according to the principle of *Increasing Precision with Decreasing Intelligence* (IPDI) (Saridis, 1989), namely: the *organization level,* performing general information processing tasks in association with a long-term memory; the *coordination level,* dealing with specific information processing tasks with a short-term memory; and the *execution level,* realizing the execution of various tasks through hardware, using feedback control methods (Fig. 1). A mathematical formulation for the organization level of intelligent machines has been proposed in Saridis and Valavanis (1988). A survey of recent advances in the theory of intelligent machines is given by Saridis (1988a).

The coordination level of intelligent machines is an intermediate structure serving as an interface between the organization level and execution level for dispatching organizational information to execution devices. It deals with real-time information of the world by generating a proper sequence of subtasks pertinent to the execution of the requested job. The purpose of this paper is to develop an analytical model for the coordination level of intelligent machines, which, with the mathematical formulation for the organization level and the well-developed control theory for the execution level, would

* Received 24 June 1989; revised 17 September 1989; received in final form 17 September 1989. The original version of this paper was presented at the 1990 IFAC World Congress which was held in Tallinn, Estonia, U.S.S.R. during August, 1990. The Published Proceedings of this IFAC Meeting may be ordered from: Pergamon Press plc, Headington Hill Hall, Oxford OX3 0BW, U.K. This paper was recommended for publication in revised form by Editor A. P. Sage.

† NASA Center for Intelligent Robotic Systems for Space Exploration and the Department of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute, Troy, New York 12180-3590, U.S.A.

‡ Author to whom all correspondence should be addressed.

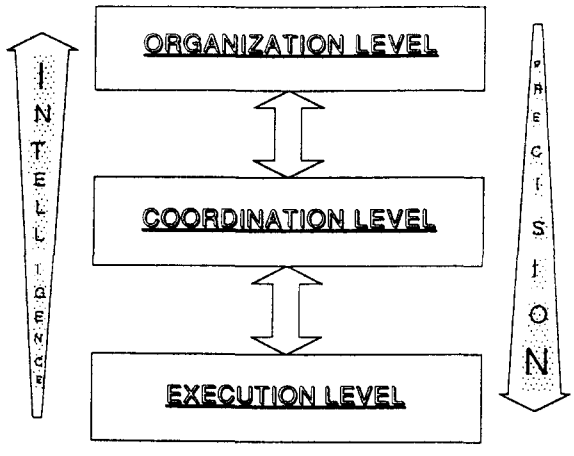

FIG. 1. The structure of intelligent machines.

complete *A Mathematical Theory for Intelligent Machines*.

Inherently, the coordination level of intelligent machines is a distributed problem-solving system (DPSS) with the achievement of coherent control and communication of various diversified processes as its key issue (Saridis, 1988b). A considerable amount of work has been done in Distributed Artificial Intelligence (DAI) during the past decade on the organization and architecture of DPSS (Decker, 1987). Multi-agent planning (Georgeff, 1984), negotiation (Smith and Davis, 1983) and the functionally-accurate, cooperative (FA/C) (Lesser and Corkill, 1981) approaches are the three major important approaches in DAI. Since the functions of subsystems in intelligent machines are deterministic, the multi-agent planning approach is used here in the architecture design of the coordination level. Various modeling tools of concurrency in computer science (Peterson, 1981; Hoare, 1985), and of discrete events in control theory (Ramadge and Wonham, 1982; Inan and Varaiya, 1988) are useful as the basic construction module of analytical models of distributed systems. This paper is focused on using a Petri net transducer to implement the *linguistic decision schemata* (Saridis and Graham, 1984) for modeling and analyzing the coordination process in the coordination level of intelligent machines.

An analytical model for the coordination level should enable us to have: (1) Formal description of individual processes within each subsystem of the coordination level; (2) Formal specification of the cooperation among subsystems; and (3) A mechanism of control and communication for task processes. The Petri net has been chosen since it can serve all these purposes quite well and also provide us with a hierarchical and modular design approach with many stepwise refinement and modification methods available. Moreover, the Petri net model offers us analytical tools for process analysis such as deadlock-freedom, liveness, boundedness, etc. and for the process evaluation such as average execution time, system utilization, etc. for the whole coordination level.

The formalism developed in this paper may also find applications in the verification of an Open System Interconnection (OSI) architecture. Some such results have been obtained for Manufacturing Message Specification (MMS) by Wang and Gildea (1988, 1989).

2. THE FRAMEWORK OF THE COORDINATION LEVEL

The topology of the coordination level can be expressed by a tree structure (D, C), where D is the root, called *dispatcher*, and C is the finite set of the subnodes, called *coordinators* (Fig. 2). It is assumed that for each coordinator there exists a bidirectional link connecting it to the dispatcher and there is no direct link between any two individual coordinators.

The dispatcher D will deal with the *control* and *communication* of the coordinators. It primarily concerns the questions of which coordinator(s) should be called for tasks (*task sharing*) and/or informed by the status of task execution (*result sharing*), given a sequence of *primitive events* (tasks) by the organizer for some specific job. The control and communication can be achieved by translating the given sequences of primitive events into the sequences of *coordinator-oriented control actions* with the necessary information and dispatching them to the corresponding coordinators at the appropriate time. The dispatcher is also responsible for

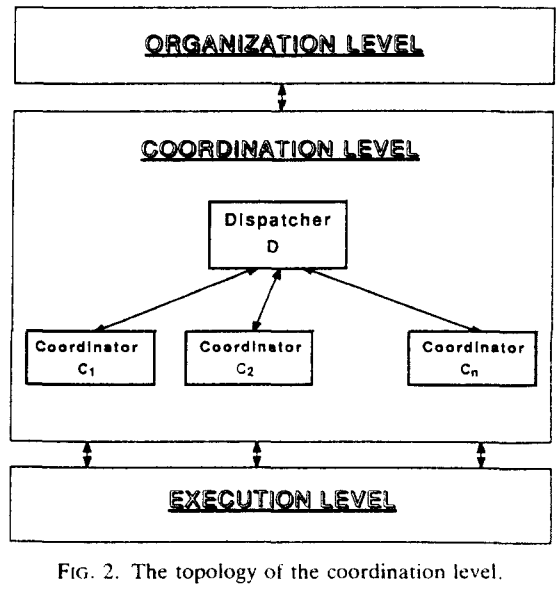

FIG. 2. The topology of the coordination level.

formulating the feedback to the organization level after the completion of the job. To this end, the dispatcher requires the following capabilities:

- A *communication facility* which allows the dispatcher to receive and send information from and to the organization level and coordinators.
- A *data processing ability* which describes the command information from the organization level and the feedback information from coordinators, updates and provides information for the decision-making units of the dispatcher.
- A *task processing ability* which identifies the task to be executed, selects the appropriate control procedures for the corresponding coordinators and formulates the feedback required by the organization level.
- A *learning ability* which improves the task processing ability of the dispatcher by reducing uncertainties in decision-making and information processing as more task execution experience is obtained.

Each coordinator is associated with a number of devices and will process the *operation* and *data transfer* of these devices. A coordinator can be considered as an expert of deterministic functions in some specific field, with the ability of selecting one among alternative actions that may accomplish the same task issued by the dispatcher in different ways, according to the constraints imposed by the workspace model and timing requirements. The operation and data transfer of the devices can be achieved by translating the given coordinator-oriented control action sequences into *real-time hardware-oriented operation* sequences with the necessary data, and sending them to the devices. The coordinator should report the result to the dispatcher after the execution of a task. Capabilities required by a coordinator are exactly the same as that for the dispatcher, but in a lower and more specific level.

Figure 3 illustrates the language (or task)

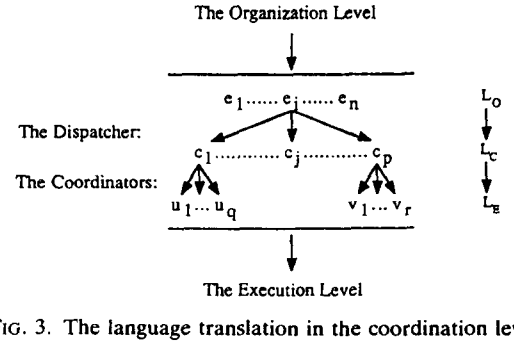

FIG. 3. The language translation in the coordination level.

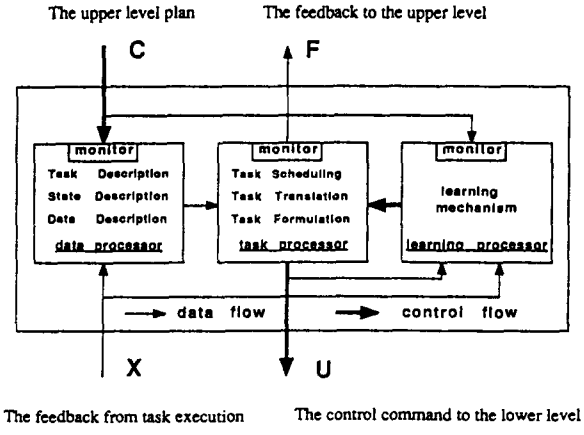

FIG. 4. A uniform architecture for dispatcher and coordinators.

translation process among the dispatcher and the coordinators. Note that the translation is achieved interactively and that the dispatcher and coordinators actually have a different time scale: one step in the dispatcher may turn out to be many steps in the coordinators. The coordinators have to cooperate under the supervision of the dispatcher in the sense that no one of them has sufficient ability and information to accomplish the entire task; mutual sharing of information is necessary to allow the dispatcher and the coordinators, as a whole, to attack the requested jobs.

The above description indicates that the dispatcher and coordinators may have identical organization at the different levels of specification. A uniform system architecture, consisting of a *data processor*, a *task processor* and a *learning processor*, is used for the dispatcher and coordinators (Fig. 4). This architecture is a direct extension of the decision module suggested by Graham and Saridis (1982).

The function of the data processor is to provide the information about the tasks to be executed and the current system status. It has been divided into three levels of description: *task description*, *state description* and *data description*. In the task description, a list of tasks to be executed from the upper level units is given. The state description presents the *preconditions* and the *postconditions* for the execution of each task and the system status in some abstract terms. In the Petri net model, the preconditions and the postconditions can be represented in terms of the input places and the output places, respectively. The data description gives the actual value for the abstract terms used in the state description. Such an information organization is very useful for the hierarchical decision-making in the task processor. The maintenance and update of the three level

descriptions is manipulated by a *monitor* based on the information from the upper level and the feedback of task execution from the lower level. The monitor is also responsible for the interconnection between the data processor and task processor.

The function of the task processor is to formulate the control commands for the lower level units. The task processor employs a hierarchical decision-making consisting of three steps: *task scheduling, task translation* and *task formulation.* Task scheduling identifies the task to be executed by checking the task description and the corresponding preconditions and post-conditions contained in the state description without referring to the actual values. If no subtasks can be executed, the task scheduling has to determine the internal operations which will make the preconditions for some tasks to become true. The task translation decomposes the task or interoperation into the control actions in an appropriate order based on the current system status. Finally, the task formulation assigns the actual data to the control actions by searching in the data description of the data processor, formulates the final complete control command, and sends it to the lower level units. With the hierarchical information description in the data processor, such a hierarchical decision-making should make the task processing fast and efficient. Upon the completion of all tasks required, a *monitor* is called to organize the feedback information to the upper level in some specified form. The monitor is also responsible for the proper interconnection with the task processor and the learning processor.

The function of the learning processor is to improve the performance of the task processor and to reduce the uncertainty in decision-making and information processing. Information used by the learning processor is indicated in Fig. 4. Various learning mechanisms can be employed by the learning processor to achieve its function. A simple linear refinement stochastic learning algorithm is used for task translation in Section 5.

The fact that the dispatcher and coordinators have identical system architecture, but at different levels of specification (or abstraction), also indicates that the coordination level has a *nested tree* topology (Meystel, 1986). This nested tree topology can be extended further to include the execution level.

The connection among the dispatcher and coordinators will be specified in terms of a Petri net derived from the coordination structure given in Section 4.

3. THE PETRI NET TRANSDUCER AND SYNCHRONOUS COMPOSITION

The previous section reveals that the basic function of the coordination level can be viewed as the translation of the high level command language issued by the organizer to the low level operation language executed by devices; therefore, a Petri net transducer has been developed as the basic construction module for the coordination level (Wang and Saridis, 1988).

A petri net N is a 4-tuple $N = (P, T, I, O)$ consisting of a finite set of places P, a finite set of transitions T, an input function I and an output function O. The set of places represents the system's states and the set of transitions represent events which can occur to change the state of the system. The input function specifies the preconditions for each event to occur and the output function describes the effects of the occurrence of each event. A place can contain a non-negative integer number of tokens. The state of the system modeled by a Petri net is given by its marking, i.e. the number of tokens in each of its places. For details of Petri net theory, a reader is referred to Peterson (1981).

The following notations are used in the sequel. $\delta(\mu, t)$ is the next-state function which gives the new marking of a Petri net N after firing a transition t under the marking μ. $R(N, \mu)$ [or $R(\mu)$, when N is clear] is the reachability set of N with the initial marking μ. Two transitions t_1 and t_2 are said to be in *parallel* with respect to μ iff $I(t_1) + I(t_2) \leq \mu$, and in *conflict* with respect to μ iff $I(t_1) \leq \mu$ and $I(t_2) \leq \mu$ but $I(t_1) + I(t_2) > \mu$. By the execution rule of Petri net, two transitions in parallel can be fired simultaneously; however, only one of the two transitions in conflict can be fired and its firing will disable the other. A Petri net is *live* with respect to μ if, for any marking in $R(\mu)$, it is possible to fire any transition in the net either immediately or after firing a sequence of other transitions. Liveness guarantees the absence of deadlocks. A Petri net is *bounded* with respect to μ if there exists a finite number k such that for any marking in $R(\mu)$ the number of tokens in each place of the net under that marking is less than or equal to k. When $k = 1$, the net is *safe.*

Definition 1. A *Petri net transducer* (PNT), M, is a 6-tuple, $M = (N, \Sigma, \Delta, \sigma, \mu, F)$, where

(i) $N = (P, T, I, O)$ is a *Petri net* with the *initial marking* μ;
(ii) Σ is a finite *input alphabet*;
(iii) Δ is a finite *output alphabet*;
(iv) σ is a *translation mapping* from $T \times (\Sigma \cup \{\lambda\})$ to finite sets of Δ^*; and
(v) $F \subset R(\mu)$ is a set of *final markings.*

Here A^* denotes the set of strings over A and λ, the empty string.

There are three parts to a PNT: an *input tape*, a *Petri net controller* and an *output tape* (Fig. 5). A *configuration* of PNT M is defined to be a triple (m, x, y) where $m \in R(\mu)$ is the current marking of N; $x \in \Sigma^*$ is the input string remaining on the input tape; $y \in \Delta^*$ is the output string emitted up to this point. A *move* by M is reflected by a binary relation $\Rightarrow_M$ (or $\Rightarrow$, when M is clear) on configurations. Specifically, for all $m \in R(\mu)$, $t \in T$, $a \in \Sigma \cup \{\lambda\}$, $x \in \Sigma^*$ and $y \in \Delta^*$ such that $\delta(m, t)$ is defined and $\sigma(t, a)$ contains $z \in \Delta^*$, we write

$$(m, ax, y) \Rightarrow (\delta(m, t), x, yz).$$

The *transitive* and *reflexive* closure of $\Rightarrow$ is denoted by $\Rightarrow^*$. Note PNTs defined here are nondeterministic in both the firing of transitions and the emitting of output strings. The *translation* defined by M is the set $\tau(M) = \{(x, y) \mid (\mu, x, \lambda) \Rightarrow^* (m, \lambda, y)$ for some $m \in F\}$ The *input language* and the *output language* of M are

$$\alpha(M) = \{x \mid (x, y) \in \tau(M) \text{ for some } y \in \Delta^*\} \text{ and}$$

$$\omega(M) = \{y \mid (x, y) \in \tau(M) \text{ for some } x \in \Sigma^*\},$$

respectively.

A PNT *halts* at configuration (m, ax, y) when no transition for which $\sigma(t, a)$ is defined are enabled at the marking m. We call m the *deadlock marking* of the PNT. When a deadlock marking occurs, the input string will be rejected. Note that a deadlock marking of PNT is not necessarily a deadlock marking of its Petri net.

For any $x = x_1a_1a_2x_2 \in \Sigma^*$ with $(\mu, x, \lambda) \Rightarrow^* (m, a_1a_2x_2, y)$, symbols a_1 and a_2 are said to be *parallel* (or *in conflict*) iff there exist t_1 and t_2 in T such that both $\sigma(t_1, a_1)$ and $\sigma(t_2, a_2)$ are defined and t_1 and t_2 are parallel (or in conflict) with respect to the marking m. Two symbols in parallel can be translated simultaneously; however, only one of the two symbols in conflict can be translated and its translation will disable the translation of the other.

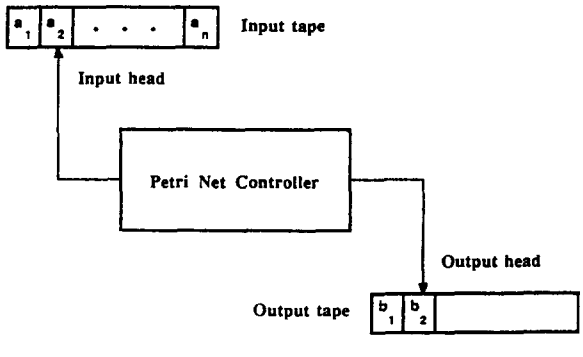

FIG. 5. The Petri net transducer (PNT).

To verify the validity of PNT being a consistent model for the dispatcher and coordinators, the possibility that the output language of some PNT cannot be translated further by any PNT has to be excluded. Let us consider a special class of PNTs, called *Simple PNT* (SPNT), with the property that for any $t \in T$ there exists one and only one $a \in \Sigma \cup \{\lambda\}$ such that $\sigma(t, a)$ is defined. The following theorem indicates the importance of this type of PNTs:

Theorem 1. For any PNT M, there exists a SPNT M′ such that $\tau(M') = \tau(M)$.

Based on this result, it is easy to show the language property of PNT can be characterized by:

Theorem 2. The input and output languages of a PNT are both Petri net languages (PNL).

This theorem guarantees that PNT can be used as a consistent model for the dispatcher and coordinators. The proofs for all the theorems in this paper are given in Wang and Saridis (1989).

The synchronous composition of PNTs is introduced to describe and specify the cooperation among the coordinators in task processing. The *project* of a string x on a language L is defined by

$$\lambda \uparrow_L = \lambda, \quad a(xa) \uparrow_L = \begin{cases} \text{undefined} & \text{if } x \uparrow_L \notin \bar{L} \\ (x \uparrow_L)a & \text{if } a \in \Sigma \\ x \uparrow_L & \text{if } a \notin \Sigma \end{cases}$$

where $s \uparrow_L$ denote the project of s on language L, Σ is the alphabet of L, and $\bar{L}$ is the closure of L.

Definition 2. The *synchronous composition* of two PNTs $M_i = (N_i, \Sigma_i, \Delta_i, \sigma_i, \mu_i, F_i)$, $i = 1, 2$, is a PNT, denoted by $M = M_1 \| M_2$. A move by M is defined to be

$$((m_1, m_2), ax, y) \Rightarrow \begin{cases} ((\delta(m_1, t_1), m_2), x, yz_1) & \text{if } a \in \Sigma_1 - \Sigma_2 \\ ((m_1, \delta(m_2, t_2)), x, yz_2) & \text{if } a \in \Sigma_2 - \Sigma_1 \\ ((\delta(m_1, t_1), \delta(m_2, t_2)), x, yz_1z_2 \text{ or } yz_2z_1) & \text{if } a \in \Sigma_1 \cap \Sigma_2 \end{cases}$$

where $z_1 \in \sigma_1(t_1, a)$ and $z_2 \in \sigma_2(t_2, a)$.

That is, the input symbols in the alphabet of one

PNT but not the alphabet of the other are translated by that PNT alone, and the input symbols in the alphabet of both PNTs are translated by two PNTs simultaneously in an arbitrary order. A string x can be translated by $M = M_1 \| M_2$ if $((\mu_1, \mu_2), x, \lambda) \Rightarrow^* ((m_1, m_2), \lambda, y)$ with $m_1 \in F_1$ and $m_2 \in F_2$.

By doing induction on the length of a string, we can prove the following important result:

$$\alpha(M_1 \| M_2) = \begin{cases} \alpha(M_1) \| \alpha(M_2) & \text{when } \Sigma_1 \cap \Sigma_2 = \phi \\ \{x \mid x \uparrow_{\alpha(M_1)} \in \alpha(M_1) \text{ and } x \uparrow_{\alpha(M_2)} \in \alpha(M_2)\} & \text{when } \Sigma_1 \cap \Sigma_2 \neq \phi \end{cases}$$

where the operator "||" on the right side of the equation is the *concurrent operator* of two languages defined in formal languages. The synchronous composition can be extended to more than two PNTs by defining

$$M_1 \|M_2\| \cdots \|M_{k-1}\| M_k = (M_1 \|M_2\| \cdots \|M_{k-1})\| M_k.$$

4. THE COORDINATION STRUCTURE AND PROCESS PROPERTIES

Now we introduce the formal model for the coordination level of intelligent machines.

Definition 3. A *coordination structure*, CS, is defined to be a 7-tuple,

$$CS = (D, C, F, R_D, S_D, R_C, S_C) \text{ where}$$

(i) $D = (N_d, \Sigma_0, \Delta_0, \sigma_d, \mu_d, F_d)$ is a PNT, the *dispatcher*, with $N_d = (P_d, T_d, I_d, O_d)$.

(ii) $C = \{C_1, C_2, \ldots, C_n\}$ is the set of *coordinators*, $n \geq 1$. Each coordinator is a SPNT $C_i = (N_c^i, \Sigma_c^i, \Delta_c^i, \sigma_c^i, \mu_c^i, F_c^i)$ with $N_c^i = (P_c^i, T_c^i, I_c^i, O_c^i)$. Define $T_c = \bigcup_{i=1}^n T_c^i$ and $P_c = \bigcup_{i=1}^n P_c^i$.

(iii) $F = \bigcup_{i=1}^n \{f_I^i, f_{SI}^i, f_O^i, f_{SO}^i\}$ is the set of *connection points*: f_I^i, f_{SI}^i, f_O^i, and f_{SO}^i are called the *input point*, *input semaphore*, *output point* and *output semaphore* of C_i, respectively.

(iv) The *dispatcher receiving mapping* R_D and *sending mapping* S_D are mappings from T_d to subsets of F. R_D and S_D satisfy the following connection constraints: $(t, f_I^i) \in S_D \Leftrightarrow (t, f_{SI}^i) \in R_D$, $(t, f_O^i) \in R_D \Leftrightarrow (t, f_{SO}^i) \in S_D$; $(t, f_I^i) \notin R_D$, $(t, f_{SO}^i) \in R_D$, $(t, f_O^i) \notin S_D$, $(t, f_{SI}^i) \notin S_D$; if $(t, f_O^i) \in R_D$ then t is not initially enabled and in any firing sequence enabling t, the number of t' with $(t', f_I^i) \in S_D$ is greater than the number of t'' with $(t'', f_O^i) \in R_D$ by one (C_i is activated sufficient times before t receiving result from it); for any f_I^i and f_O^i, there exists $t, t' \in T_d$ such that $(t, f_I^i) \in S_D$ and $(t', f_O^i) \in R_D$.

(v) The *coordinator receiving mapping* R_C and *sending mapping* S_C are mappings from T_c to subsets of F. R_C and S_C satisfy the following connection constraints:

$$(t, f_O^i) \in S_C \Leftrightarrow (t, f_{SO}^i) \in R_C \quad \text{and} \quad (t, f_{SI}^i) \in S_C;$$
$$(t, f_O^i) \notin R_C, (t, f_{SI}^i) \notin R_C, (t, f_I^i) \notin S_C, (t, f_{SO}^i) \notin S_C.$$

The configuration of coordination structure is shown in Fig. 6. The notation of connection point is similar to the concept of *port* in network theory. The connection constraints indicate that each coordinator is connected with D bidirectionally and that D can issue tasks to a coordinator only if there is a token in the corresponding input semaphore (i.e. the coordinator is available for the task) and a coordinator can report the execution result to D only if there is a token in the corresponding output semaphore (i.e. the communication facility is ready for information transferring).

Various complex connection patterns among the dispatcher and coordinators can be specified by designing different receiving and sending mappings. One of the basic connection patterns can be defined to be: (i) C_i only accesses its own connection points; (ii) there is only one initially enabled transition in C_i and it takes input from the input point; (iii) only one transition in C_i sends information to its output point. A *CS* with this type of connection pattern is called a *simple coordination structure*. We will concentrate on the simple coordination structure in this paper.

The behaviours of the dispatcher and the coordinators are specified by their *transition firing sequence* sets. As in program verification, these sequence sets can be used for designing, analyzing, implementing and simulating the models for the dispatcher and coordinators. In order to process the tasks from the dispatcher by C_i, we must have the following relationship:

$$\alpha(C_i) \supseteq \cup \{\sigma_d(t, a) \uparrow_{(T_d)^*} \mid t \in T_d^i \text{ and } a \in \Sigma_0\},$$
$$T_d^i = \{t \mid t \in T_d \text{ and } (t, f_I^i) \in S_D\}, i = 1, \ldots, n$$

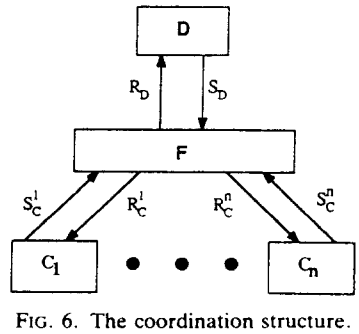

FIG. 6. The coordination structure.

which means C_i should be capable of processing all the possible task strings issued by the dispatcher. This relationship is guaranteed to be able to be satisfied by the closure property of PNL under the union operation and Theorem 2.

The operation of a *CS* can be described in terms of the *Petri net underlying CS*,

$$N = (P, T, I, O), \quad \text{where } P = P_d \cup P_c \cap F, \quad T = T_d \cup T_c,$$

$$I(t) = \begin{cases} I_d(t) \cup \{f \mid (t, f) \in R_D\} & \text{if } t \in T_d \\ I_c^i(t) \cup \{f \mid (t, f) \in R_C\} & \text{if } t \in T_c^i \end{cases},$$

$$O(t) = \begin{cases} O_d(t) \cup \{f \mid (t, f) \in S_D\} & \text{if } t \in T_d \\ O_c^i(t) \cup \{f \mid (t, f) \in S_C\} & \text{if } t \in T_c^i \end{cases}.$$

The initial marking of N is:

$$\mu(p) = \begin{cases} \mu_d(p) \text{ or } \mu_c^i(p) \text{ for } p \in P_d \text{ or } p \in P_c^i \\ 1 \text{ for } p = f_{SI}^i \text{ or } f_{SO}^i \\ 0 \text{ otherwise.} \end{cases}$$

To start operation, a *CS* receives a string (task plan) from the organizer, puts it on the input tape of the dispatcher D, and begins the process of translating (or dispatching). Once a transition t of D with $f_I^{i_1}, \ldots, f_I^{i_s}$ as its output places in F is fired with respect to the current marking of the underlying Petri net N to execute a primitive event a, it will send the selected control string $z \in \sigma_d(t, a)$ to the coordinators $C_{i_1}, \ldots, C_{i_s}$, and activate the synchronous composition $C_{i_1} \| \cdots \| C_{i_s}$. Upon completion of the task by a coordinator C_j, it will displace a token (feedback) to f_O^j. If it is enabled with respect to the current marking of N at the time, the feedback will be taken by the dispatcher to continue the process, and C_j will become idle again. Once the dispatcher reaches its final marking and the coordinators are either in the initial marking (i.e. no task processing for a while) or the final marking, the entire task process is completed successfully.

A string $s \in \Sigma_O^*$ is said to be executable by the *CS* if $(\mu_d, s, \lambda) \Rightarrow^* (m, \lambda, y)$, $m \in F_d$ and the final configuration of each of coordinators is either $(\mu_c^i, \lambda, y_c^i)$ or (m_c^i, λ, y_c^i), $m_c^i \in F_c^i$. It should be pointed out that not every string in $\alpha(D)$ is executable by the *CS*, because the additional connection restrictions imposed by the receiving and sending mappings. However, we can show that a transition enabled in N_d after finite step firing can also be enabled in N after the same steps of firing (generally by a different path, however). In any case, it has to be guaranteed that every string (or task plan) in the set of task plans L_0 (a subset of $\alpha(D)$) should be executable by *CS* during its design phase.

Clearly, the synchronous composition provides the dispatcher with a mechanism to synchronize the task execution of the coordinators and the Petri net N specifies the precedence relation among the activities in the dispatcher and coordinators and therefore defines the *information structure* of the CS. From the point of view of execution of N, a string issued by the organization level can be considered as a path specification in the N_d, and, in turn, strings selected by transitions of D can be thought of as the path specifications in the Petri nets of the coordinators.

The underlying Petri net N also enables us to use the Petri net concepts and analysis methods to study the process properties of the coordination level, such as liveness, boundedness, reversibility, consistency, repetitiveness, etc. The following theorem presents the results about the boundedness and liveness of the coordination structure, which guarantee the structural stability of the structure and the absence of deadlock in the coordination.

Theorem 3. The Petri net N underlying CS is bounded (live) if all the Petri nets of the dispatcher and coordinators are bounded (live).

For the construction of coordination structures, the methods of building the bounded and live Petri net models for manufacturing systems would be very useful. The stepwise refinement approaches developed in Valette (1979) and Suzuki and Murata (1983) can be easily adapted for PNTs.

5. DECISION MAKING IN THE COORDINATION STRUCTURE

The decision-making in the coordination level is achieved through two steps: task scheduling and task translation. Task scheduling is the process of identifying the appropriate tasks to be executed for the requested job. Once a task is located, task translation takes place by decomposing the task into a subtask sequence and, after being assigned with real-time information (task formulation), executing subtasks or sending them to the corresponding infimal units. In terms of PNT, the problem of task scheduling and translation for a given task a is to find an enabled t such that $\sigma(t, a)$ is defined, and then select the right translation string from $\sigma(t, a)$ for the transition t.

A simple and uniform scheduling procedure can be designed based on the execution rule of Petri net. Let $M = (N, \Sigma, \Delta, \sigma, \mu, F)$ be a PNT representing the dispatcher or a coordinator. For any $a \in \Sigma$, we define $T(a) = \{t \mid \sigma(t, a)$ is

defined} and $T_\lambda = T(\lambda) = \{t \mid \sigma(t, \lambda)$ is defined} (the set of *internal operations*). Let Q_T and Q_D be two queues, Q_T store the unexecuted tasks and Q_D the tasks delayed for execution due to their processing transitions not being enabled at the appropriate time. Function F(Q) deletes and returns the first element of Q, I(Q, a), inserts a to Q at the end, U(Q_1, Q_2) unifies Q_1 and Q_2 by placing Q_2 at the end of Q_1, and N(Q) empties Q. Let $v = a_1 a_2 \ldots a_s \in \Delta^*$ be a task string to be executed; the scheduling procedure for M can be described as:

Scheduling Procedure (SP):

(1) $Q_T := \{a_1, a_2, \ldots, a_s\}$, $Q_D := \phi$;
(2) IF Q_T is empty THEN STOP;
(3) $u := F(Q_T)$;
(4) IF there exists a $t \in T(u)$ and t is enabled THEN firing t, GOTO 7;
(5) IF there exists an internal operation sequence $e \in T_\lambda^*$ such that $t \in T(u)$ is enabled by firing e THEN firing et, GOTO 7;
(6) I(Q_D, u), IF Q_T is empty THEN $Q_T := Q_D$ and N(Q_D), GOTO 2;
(7) IF Q_D is not empty THEN $Q_T :=$ U(Q_D, Q_T) and N(Q_D), GOTO 2.

In SP, each task is examined in the order appearing in the task string v. If a task is executable at the time, it will be executed right away. Otherwise an effort is made to find a sequence of internal operations which will enable the task. If the effort fails, the task will be removed from Q_T and added to Q_D. Once there is a change in the state of M, all the delayed tasks in Q_D will be moved back to Q_T in their original order and be examined again, since it is desired to keep the task order as specified as much as possible. For a large and complex PNT, the heuristic search algorithm (Passino and Antsaklis, 1988) may be used to find the internal operation sequence. For a bounded, average size PNT, however, the simple breadth-first search along the reachability tree of PNT by firing only the internal operations under the current marking can serve the purpose quite well. The SP will terminate finitely since it is assumed that the task strings issued are compatible and complete.

Task translation can be achieved in either an active or a passive fashion. In the active approaches, the translation of a task is accomplished on-line based on a set of rules and a data base which describes the related environment and system status information. In the passive approaches, a fixed number of translations for a task are pre-specified and the translating is to select one of them according to the current situations. For a PNT, the way of translating is indicated by how the translation mapping σ is generated. We will consider the passive approach here by assuming a fixed number of translations available for each transition, and use the learning algorithm to learn the best translation for a task in a particular situation.

Let t be a transition of a PNT $M = (N, \Sigma, \Delta, \sigma, \mu, F)$. The number of translations designed for t is $M_t = \Sigma\,|\sigma(t, a)|$. Let x_t represent the information about the system status and temporal constraints provided in the input places of t (token color) and $u_t \in U_t \equiv \{a \in \Sigma \cup \{\lambda\} \mid \sigma(t, a)$ is defined} represent a task to be translated by t. A *situation* is defined to be a combination of x_t and u_t, i.e., (u_t, x_t). The number of situation distinguished by t is designated as N_t. Now consider a matrix of subjective properties, $(p_{ij})_{Mt \times Nt}$. The decision rule of the probabilistic method for choosing a translation is:

Decision Rule (DR): When situation $(u_t, x_t)_j$ is observed, choose a translation s_i using a random strategy with the subjective probability p_{ij}, $i = 1, \ldots, M_t$.

A random performance index is associated with each translation. After the execution of the action specified by s_i for situation $(u_t, x_t)_j$, update the performance estimate using the algorithm:

$$\tilde{J}_{ij}(k_{ij} + 1) = \tilde{J}_{ij}(k_{ij}) + \beta(k_{ij} + 1) \times [J_{obs}(k_{ij} + 1) - \tilde{J}_{ij}(k_{ih})]$$

where J_{obs} is the observed performance value, $\tilde{J}$ the performance estimate, and k_{ij} the number of times the event $((u_t, x_t)_j s_i)$ has occurred.

After updating the performance estimate, update the subjective probabilities by the algorithm:

$$p_{ij}(k + 1) = p_{ij}(k) + \gamma(k + 1)[\xi_{ik}(k) - p_{ij}(k)],$$

where $\xi_{ij}(k) = 1$ if $\tilde{J}_{ij} = \min_l \tilde{J}_{lj}$, $\xi_{ij}(k) = 0$ otherwise.

When $\beta(k_{ij})$ and $\gamma(k)$ satisfy Dvoretsky's convergence condition, we have

$$\text{Prob}\{\lim_{k \to \infty} [\tilde{J}_{ij}(k) - \bar{J}_{ij}] = 0\} = 1,$$

$$\text{Prob}[\lim_{k \to \infty} p_{ij}(k) = \delta_{ip}] = 1,$$

where $\bar{J}_{ij}$ is the expected value of J_{ij}, $\bar{J}_{pj} = \min_l \bar{J}_{lj}$, and $\delta_{ip} = 1$ if $i = p$ or 0 if $i \neq p$.

The proof of convergence is given in Saridis and Graham (1984).

Assuming that the initial performance estimate for a transition is available, we can find the *most conservative* initial subjective probabilities by Jaynes' *Maximum Entropy Principle* (Wang and Saridis, 1989).

6. ENTROPY MEASURE OF UNCERTAINTY AND LEARNING

The learning process can be measured by the entropy associated with the subjective probabilities. For a PNT M, its *translation uncertainty* is defined to be the total entropy of subjective probabilities assigned to the transitions of M, that is,

$$H(M) \equiv \Sigma_{t\in T} H(t) = \Sigma_{t\in T}\{H[(u_t, x_t)] + H[t/(u_t, x_t)]\}$$
$$= -\Sigma_{t\in T}\Sigma_j p_j \ln p_j - \Sigma_{t\in T}\Sigma_j p_j \Sigma_i p_{ij} \ln p_{ij},$$

where p_j is the objective probability of the situation $(u_t, x_t)_j$ occurring. Define

$$H(E) = -\Sigma_{t\in T}\Sigma_j p_j \ln p_j,$$
$$H(T/E) = -\Sigma_{t\in T}\Sigma_j p_j \Sigma_i p_{ij} \ln p_{ij},$$

then,

$$H(M) = H(E) + H(T/E).$$

The expression indicates that the translation uncertainty H(M) can be divided into two parts: the *environment uncertainty* H(E), caused by the uncertainty of the environment (include the uncertainty in task assignment); the *pure translation uncertainty* H(T/E), the uncertainty in translation with the given environment. Clearly, only the pure translation uncertainty can be reduced by learning. The uncertainty in the whole coordination structure CS is

$$H(CS) = H(E_{CS}) + H(T_{CS}/E_{CS}), \text{ where}$$
$$H(E_{CS}) = H(E_D) + \Sigma_{i=1}^{n} H(E_{Ci}) \quad \text{and}$$
$$H(T_{CS}/E_{CS}) = H(T_D/E_D) + \Sigma_{i=1}^{n} H(T_{Ci}/E_{Ci}).$$

It is clear now that the coordination structure is a Petri net implementation of the (*embedded*) *decision schema* (Saridis and Graham, 1984). The whole learning process bears the similar structure to that of the *hierarchical learning automata* (Mandyam *et al.*, 1981). For the case of the small number of situations, a learning by recording the conditional probabilities under specific situations will not cause a serious problem in memory space and learning speed. As the number of situations and tasks to be processed is increasing, however, the problem of memory space and learning speed becomes more and more serious. For the case of the larger number of situations, the pattern-recognizing learning algorithm developed by Barto and Anandan (1985), which avoids the maintenance of separate selection probabilities of each situation by parameterizing the conditional probabilities and constructing a mapping from the situations to the parameter, should be used.

In the limit case when the pure translation uncertainty of a PNT is reduced to zero, the optimal translation is found for all transitions in the PNT. However, in general, this does not necessarily imply that a global optimal translation is achieved for the PNT, simply because the optimal translations of a transition might cause worse situations for the subsequent transitions, and thus increase the total task execution cost. To achieve the global optimal translation, it is imperative to specify the influences of the translation of a transition to others, and use a learning algorithm based on global information. This is especially true for cases when the dependent tasks or events are processed by several transitions. However, in some cases, an analytical expression for such influences may be too complex to be established for the transitions with diversified functions.

The pure translation uncertainty of a PNT indicates how knowledgeable the PNT is about its task execution. Since learning from the execution can reduce the pure translation uncertainty, learning can make a PNT more knowledgeable on its task execution. This observation reveals that decision-making in the coordination level bears "dual" character: on one side, the decisions made have to make the dispatcher or coordinators to accomplish the requested task with some optimization criterion; on the other side, these decisions should also make it more knowledgeable about its task execution in the future by reducing its pure translation uncertainty. The trade-off between the two sides should be judged by some criterion.

7. A CASE STUDY

The model developed has been applied to construct the coordination level of a simple intelligent manipulator system composed of a general-purpose manipulator of six degrees of freedom with a gripper, a vision system that recognizes various objects and provides visual information about their position and orientation, and a sensor system that provides all kinds of sensory information. The coordination structure designed consists of a dispatcher, a motion coordinator, a vision coordinator and a sensor coordinator (Fig. 7, where, for the sake of brevity, the connection points are pictured together for all the coordinators).

The task plans from the Organization Level is assumed to be generated by the grammar

$$G = (N, \Sigma_0, P, S),$$

where $N = \{S, M, Q, H\}$, $\Sigma_0 = \{e_1, e_2, e_3, e_4\}$, and

$$P = \{S \rightarrow e_1 M, M \rightarrow e_2 S \mid e_2 Q, Q \rightarrow e_3 H, H \rightarrow e_4 Q \mid e_4 S \mid e_4\}.$$

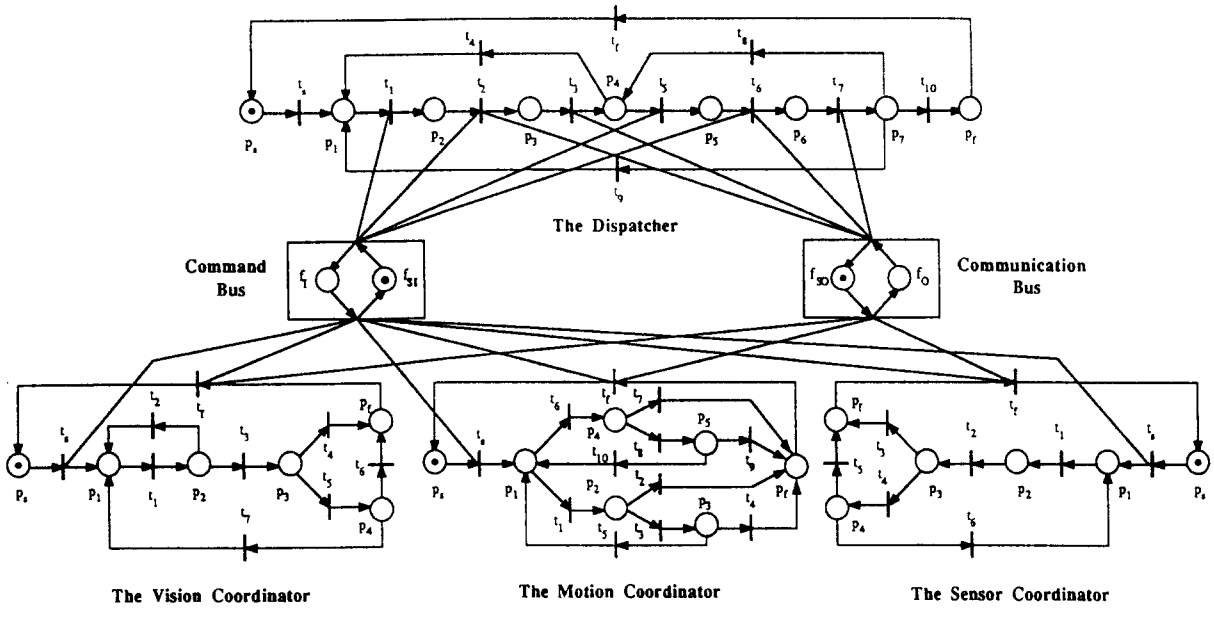

FIG. 7. The coordination structure of an intelligent manipulator system.

The primitive events e_1, e_2, e_3 are tasks involved with the vision, motion and sensor coordinators, respectively; e_4 is the task of grasping or putting down objects. The input alphabets of the vision, the motion and the sensor coordinators are $\Sigma_v = \{c, r, a_v, i, f\}$, $\Sigma_m = \{m, h, i, f\}$, and $\Sigma_s = \{s, a_s, i, f\}$, respectively.

The final markings of the dispatcher and coordinators are reached when there is a token in the corresponding place p_f. Their translation mappings are defined to be:

Dispatcher:

$$\sigma_d(t_1, e_1) = \sigma_d(t_1, \lambda) = \{\lambda, ca_v f, crca_v f\},$$
$$\sigma_d(t_2, e_2) = \{mf, (mca_v i)^n f, n > 0\},$$
$$\sigma_d(t_5, e_3) = \sigma_d(t_5, \lambda) = \{\lambda, sa_s f\},$$
$$\sigma_d(t_6, e_4) = \{gf, (gsa_s i)^n f, n > 0\}.$$

Vision: $\sigma_v(t_1, c)$ = {instructions to control camera and take picture}, $\sigma_v(t_2, r)$ = {instructions to change the lighting condition}, $\sigma_v(t_3, a_v)$ = {algorithms of image processing and analysis}, $\sigma_v(t_4, f) = \sigma_v(t_6, f)$ = {formulating the feedback information}, $\sigma_v(t_5, i)$ = {sending the visual information to the motion coordinator}.

Sensor: $\sigma_s(t_1, s)$ = {instructions to control sensors and take data}, $\sigma_s(t_2, a_s)$ = {algorithms of data processing and analysis}, $\sigma_s(t_3, f) = \sigma_s(t_5, f)$ = {formulating the feedback information}, $\sigma_s(t_4, i)$ = {sending the sensory information to the motion coordinator}.

Motion; $\sigma_m(t_1, m)$ = {algorithms of path planning and motion control for the arm}, $\sigma_m(t_3, i)$ = {receiving the visual information}, $\sigma_m(t_6, g)$ = {algorithms of fine path planning and grasping or putting down objects for the hand}, $\sigma_m(t_8, i)$ = {receiving the sensory information}, $\sigma_m(t_2, f) = \sigma_m(t_4, f) = \sigma_m(t_7, f) = \sigma_m(t_9, f)$ = {formulating the feedback information}.

All other transitions of the dispatcher and coordinators are internal operations.

Clearly, the input language of the dispatcher is exactly the task plans issued by the organizer. The empty string λ in the translation mapping σ_d means no action since the required visual or sensory information is already available. The interactive motion control strings $(mca_v i)^n f$ and $(gsa_s j)^n f$ of the dispatcher involve the synchronous composition of two coordinators, i.e. motion || vision and motion || sensor, respectively. When these control strings are issued for motion or grasp tasks, the motion and vision or sensor coordinators have to work cooperatively to achieve the tasks. That is, the arm (or hand) moves step by step with the assistance of the cameras (or sensors) until the specified position and orientation are reached.

The task simulation of picking up PC boards and inserting them into slots is performed. The task plan assigned is $e_1e_2e_3e_4e_1e_2e_3e_4$. It is assumed that there are, respectively, two algorithms for image processing and analysis, data processing and analysis, the arm path planning and motion control, and the hand path planning and grasp and insertion control. A coordinator has to learn which of the two algorithms is best for the corresponding tasks. The learning for the dispatcher is to decide the way to accomplish the arm and hand motion.

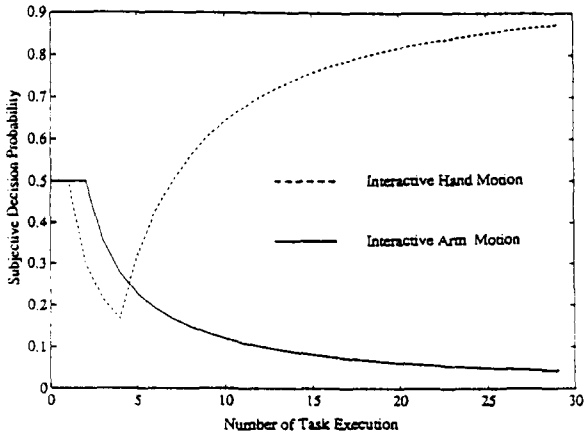

FIG. 8. The learning curves of t_2 and t_6 in the dispatcher.

When the PC boards or the slots are moving during the task process, the interactive arm motion control $(mca_v i)^n f$ should be used, otherwise mf should be the better choice. Similarly, depending on the degree of the uncertainty of the visual information provided by the vision about the PC board and slot location, either $(gsa_s i)^n f$ or gf should be the optimal approach of gripping and inserting.

The zero initial cost and the uniform initial subjective probabilities are used for all the translations in the simulation. Figure 8 gives the learning curves of transitions t_2 and t_6 in the dispatcher upon executing the task 30 times. Figure 9 gives the corresponding pure translation entropies of the dispatcher and coordinators. The entropy curves indicate clearly that the learning speeds in the coordinators are much faster than the learning speeds in the dispatcher. This is because the learning in the dispatcher depends on the learning in the coordinators and that one step of execution in the dispatcher may correspond to many steps in the coordinators.

An example with the detail specification of the situations and the performance indices for each transition of the dispatcher and coordinators is given in Wang and Saridis (1989).

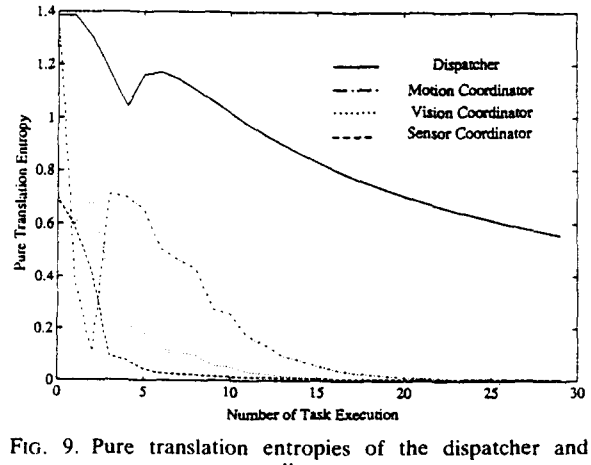

FIG. 9. Pure translation entropies of the dispatcher and coordinators.

8. CONCLUSION

A coordination theory for intelligent machines has been developed by establishing an analytical model for their coordination level. PNT plays the role of basic module for the description of task translation and task process in our model. The cooperation and connection among the dispatcher and coordinators are specified by the synchronous composition of PNTs and receiving and sending of mappings of the coordination structure. The process analysis is achieved within the context of Petri net theory since various concepts and methods have been developed for a Petri net to serve such purposes. Moreover, process evaluation, like average execution time, can be performed by using a timed Petri net. The execution rule of a Petri net provides the base for designing the task scheduling procedure and the learning algorithm gives an adaptive approach for finding the optimal task translation in the uncertain environment. This model of coordination provides an analytical mechanism of control and communication for autonomous intelligent control systems in various fields of modern industry such as intelligent robotic systems and computer integrated manufacturing systems.

REFERENCES

Albus, J. S. (1975). A new approach to manipulation control: The cerebellar model articulation controller. *Trans. ASME J. Dynam. Syst., Meas. Control*, **97**, 220–227.

Antsaklis, P. J., K. M. Passino and S. J. Wang (1988). Automonous control systems: Architecture and fundamental issues. *Proc. Amer. Control Conf.*, **1**, 602–607.

Barto, A. G. and P. Anandan (1985). Pattern-recognizing stochastic learning automata. *IEEE Trans. Syst. Man Cybern.*, **SMC-15**, 360–375.

Bejczy, A. (1986). Task driven control. *IEEE Workshop on Intelligent Control*, p. 38. Rensselaer Polytechnic Institute (RPI), Troy, New York.

Decker, K. S. (1987). Distributed problem-solving techniques: A survey. *IEEE Trans. Syst. Man Cybern.*, **SMC-17**, 729–740.

Georgeff, M. (1984). A theory of action for multi agent planning. *Proc. 9th IJCAI*, 121–125.

Graham, J. H. and G. N. Saridis (1982). Linguistic decision structures for hierarchical systems. *IEEE Trans. Syst. Man Cybern.*, **SMC-12**, 323–333.

Hoare, C. A. R. (1985). *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs, NJ.

Inan, K. and P. Varaiya (1988). Finitely recursive process models for discrete event systems. *IEEE Trans. Aut. Control*, **AC-33**, 626–639.

Lesser, V. R. and D. D. Corkill (1981). Functionally-accurate, cooperative distributed systems. *IEEE Trans. Syst. Man Cybern.*, **SMC-11**, 81–96.

Mandyam, A. L., M. A. L. Thathachar and K. R. Ramakrishnan (1981). A hierarchical system of learning automata. *IEEE Trans. Syst. Man Cybern.*, **SMC-11**, 236–241.

Meystel, A. (1986). *Nested Hierarchical Control: Theory of Team Control Applied to Autonomous Robots*. Lab. Applied Machine Intelligence and Robotics, ECE Dept, Drexel University, PA.

Pao, Y. H. (1986). Some views on analytic and artificial

intelligence approaches. *Proc. IEEE Workshop on Intelligent Control* p. 29 RPI, Troy, NY.

Passino, K. M. and P. J. Antsaklis (1988). Planning via heuristic search in a Petri net framework. *Proc. 3rd IEEE Int. Intelligent Control Symp*, Arlington, VA.

Peterson, J. L. (1981). *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs, NJ.

Ramadge, P. J. and W. M. Wonham (1982). Supervision of discrete event processes. *Proc. 21st IEEE Conf. on Decision and Control*, 1228–1229.

Saridis, G. N. (1977). *Self-organization Controls of Stochastic Systems*. Marcel Dekker, N.Y.

Saridis, G. N. (1983). Intelligent robotic control. *IEEE Trans. Aut. Control*, **AC-28,** 547–557.

Saridis, G. N. (1986). Foundations of intelligent controls. *Proc. IEEE Workshop on Intelligent Control*, pp. 23–27. RPI, Troy, NY.

Saridis, G. N. (1988a). On the theory of intelligent machines: A survey. *Proc. 27th IEEE Conf. on Decision and Control*, Austin, Texas, 1799–1804.

Saridis, G. N. (1988b). Intelligent Machines: distributed vs hierarchical intelligence. *Proc. IFAC/IMACS Symp. on Distributed Intelligence Syst.*, Varna, Bulgaria, 34–39.

Saridis, G. N. (1989). Analytic formulation of the principle of increasing precision with decreasing intelligence[4] for Intelligent Machines. *Automatica*, **25,** 461–467.

Saridis, G. N. and J. H. Graham (1984). Lingusitic decision schemata for intelligent robots, *Automatica*, 121–126.

Saridis, G. N. and K. P. Valavanis (1988). Analytic design of Intelligent Machines. *Automatica*, **24,** 123–133.

Smith, R. D. and R. Davis, (1983). Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, **20,** 63–109.

Stephanou, H. E. (1986). Knowledge based control systems. *Proc. IEEE Workshop on Intelligent Control*, p. 116– RPI, Troy, NY.

Suzuki, I. and T. Murata (1983). A method for stepwise refinements and abstractions of Petri nets. *J. Comput. Syst. Sci.*, **27,** 51–76.

Valette, R. (1979). Analysis of Petri Nets by stepwide refinements. *J. Comput. Syst. Sci.*, **18,** 35–46.

Vamos, T. (1987). Metalanguages—Conceptual model: Bridge between machine and human intelligence. *Proc. 1st Int. Symp. on AI and Expert Syst.*, 237–287.

Wang, F. Y. and K. Gildea (1988). MMS design and implementation using Petri nets, DOC#CIMMN88TR178. CIM Program, Center for Manufacturing Productivity and Technology Transfer, RPI, Troy, N.Y. Also (1990), *Proc. First Int. Workshop on Formal Methods in Engng Design, Manuf. and Assembly*, Colorado Springs, CO, pp. 184–201.

Wang, F. Y. and K. Gildea (1989). A colored Petri net model for connection management services in MMS. *Comput. Communic. Rev.*, **19,** 76–98.

Wang, F. Y. and G. N. Saridis (1988). A formal model for coordination of Intelligent Machines using Petri nets. *Proc. 3rd IEEE Int. Intelligent Control Symp.*, Arlington, VA, pp. 28–33.

Wang, F. Y. and G. N. Saridis (1989). *A Coordination Model for Intelligent Machines*, TR#14. Center of Intelligent Robotic Systems for Space Exploration, RPI, Troy, NY.