AGGREGATION AND MULTI-LEVEL CONTROL IN DISCRETE EVENT DYNAMIC SYSTEMS[1]

Cüneyt M. Özveren[2]

Alan S. Willsky[2]

August 7, 1989

Abstract

In this paper we consider the problem of higher-level aggregate modelling and control of discrete-event dynamic systems (DEDS) modelled as finite state automata in which some events are controllable, some are observed, and some represent events to be tracked. The higher-level models considered correspond to associating specified sequences of events in the original system to single macroscopic events in the higher-level model. We also consider the problem of designing a compensator that can be used to restrict microscopic behavior so that the system will only produce strings of these primitive sequences or tasks. With this lower level control in place we can construct higher-level models which typically have many fewer states and events than the original system. Also, motivated by applications such as flexible manufacturing, we address the problem of constructing and controlling higher-level models of interconnections of DEDS. This allows us to "slow down" the combinatorial explosion typically present in computations involving interacting automata.

[1]Research supported by the Air Force Office of Scientific Research under Grant AFOSR-88-0032 and by the Army Research Office under Grant DAAL03-86-K0171.

[2]Laboratory for Information and Decision Systems, MIT, Cambridge, MA 02139.

1 Introduction

The study of complex systems has frequently prompted research on tools for aggregation and multi-level analysis. In this paper, we study such tools in the context of Discrete Event Dynamic Systems (DEDS). DEDS have been studied extensively by computer scientists, and recently, the notion of control of a DEDS has been introduced by Wonham, Ramadge, et al. [1,7,8,9]. This work assumes a finite state model in which certain events in the system can be enabled or disabled. The control of the system is achieved by choice of control inputs that enable or disable these events. We also consider a similar model in our work.

A major issue of concern in the work of Wonham and Ramadge, as well as other work on DEDS is that of computational complexity, and the goal of this paper is to address certain issues of complexity. In particular, in many applications the desired range of behavior of a DEDS is significantly smaller and more structured than its full range of possible behaviors. For example, a workstation in a flexible manufacturing system may have considerable flexibility in the sequence of operations it performs. However, only particular sequences correspond to useful tasks. This idea underlies the notion of a legal language introduced in [8]. In the analysis described in this paper we use it as well to develop a method for higher-level modelling and control in which a sequence of events corresponding to a task is mapped to a single macro-event at the higher-level. Also, in DEDS described as interconnections of subsystems the overall state space for the entire system can be enormous. However, in many applications, such as a flexible manufacturing system consisiting of interconnections of workstations, the desired coordination of the subsystems is at the task level, and thus we can consider the interactions of their individual aggregate models. These

higher-level characterizations allow us to represent sets of states by a single state and sets of strings by a single event. We thus achieve both spatial and temporal aggregation that can greatly reduce the apparent computational explosion arising in the analysis of extremely complex systems.

The work described in this paper builds on several of our previous papers [2,3,4, 5,6] and can be viewed as the culmination of an effort to develop a regulator theory for DEDS. As we will see, our development will involve controlling the system so that its behavior is restricted to completing the desired tasks. To address this, we will rely on the notions of tracking and restrictability that we developed in [5]. The latter of these is closely related to the notion of constraining behavior to a legal language. However, by describing the desired behavior in terms of primitive tasks, we achieve significant efficiencies, and through the use of the notion of eventual restrictability we are able to directly accomodate the phenomenon of set-up, i.e., the externally irrelevant transient behavior arising when one switches between tasks.

We will see that many of the components of our work are relevant here. In particular, our notion of stability [6], i.e., of driving the system to a specified set of states is central to most of our constructions. Furthermore, since we assume a model in which only some events are observed, we will need to make use of our results in observability [3] and output stabilization [4]. Finally, in order to derive an upper-level model, it will be necessary to be able to use our observations to reconstruct the sequence of tasks that has been performed. This is closely related to the problem of invertibility stated in [2].

In the next section, we introduce the mathematical framework considered in this paper and summarize those parts of our previous work that will be used in the sequel.

In Section 3, we formulate a notion of modelling based on a given set of macro-events or primitives, which allows us to characterize higher-level models of DEDS. In Section 4, motivated by flexible manufacturing systems, we define tasks as primitives, introduce notions of reachability and observability of tasks, and construct task compensators and detectors. Using these components, we construct an overall task-level control system which accepts task requests as input and controls the system to achieve the desired sequence. This leads to a simple higher-level model whose transitions only involve the set-up and completion of tasks. In Section 5, we show how a system composed of m subsystems can be modelled by a composition of the higher-level models of each subsystem. Also, we illustrate our approach using a simple manufacturing example, and show how the overall control of the task sequence of this system can be achieved by a higher-level control acting on the composite task-level model. Finally, in Section 6, we summarize our results and discuss several directions for further work.

2 Background and Preliminaries

2.1 System Model

The class of systems we consider are nondeterministic finite-state automata with intermittent event observations. The basic object of interest is the quintuple:

$$G = (X, \Sigma, \Phi, \Gamma, \Xi) \tag{2.1}$$

where X is the finite set of states, with $n = |X|$, Σ is the finite set of possible events, $\Phi \subset \Sigma$ is the set of controllable events, $\Gamma \subset \Sigma$ is the set of observable events, and $\Xi \subset \Sigma$ is the set of tracking events. Also, let $U = 2^{\Sigma}$ denote the set of admissible control inputs consisting of a specified collection of subsets of Σ. The dynamics defined on G are as follows, where $\overline{\Phi}$ denotes the complement of Φ:

$$x[k+1] \in f(x[k], \sigma[k+1]) \tag{2.2}$$

$$\sigma[k+1] \in (d(x[k]) \cap u[k]) \cup (d(x[k]) \cap \overline{\Phi}) \tag{2.3}$$

Here, $x[k] \in X$ is the state after the kth event, $\sigma[k] \in \Sigma$ is the $(k+1)$st event, and $u[k] \in U$ is the control input after the kth event. The function $d : X \rightarrow 2^{\Sigma}$ is a set-valued function that specifies the set of possible events defined at each state (so that, in general, not all events are possible from each state), and the function $f : X \times \Sigma \rightarrow X$ is also set-valued, so that the state following a particular event is not necessarily known with certainty. We assume that $\Phi \subset \Gamma$. This assumption simplifies the presentation of our results, but it is possible to get similar results, at a cost of additional computational complexity, if it is relaxed.

Our model of the output process is quite simple: whenever an event in Γ occurs, we observe it; otherwise, we see nothing. Specifically, we define the output function

$h : \Sigma \rightarrow \Gamma \cup \{\epsilon\}$, where ϵ is the "null transition", by

$$h(\sigma) = \begin{cases} \sigma & \text{if } \sigma \in \Gamma \\ \epsilon & \text{otherwise} \end{cases} \tag{2.4}$$

Then, our output equation is

$$\gamma[k+1] = h(\sigma[k+1]) \tag{2.5}$$

Note that h can be thought of as a map from Σ^* to Γ^*, where Γ^* denotes the set of all strings of finite length with elements in Γ, including the empty string ϵ. In particular, $h(\sigma_1 \cdots \sigma_n) = h(\sigma_1) \cdots h(\sigma_n)$.

The set Ξ, which we term the tracking alphabet, represents events of interest for tracking purposes. This formulation allows us to define tracking over a selected alphabet so that we do not worry about listing intermediary events that are not important in tracking. We use $t : \Sigma^* \rightarrow \Xi^*$, to denote the projection of strings over Σ into Ξ^*. The quintuple $A = (G, f, d, h, t)$[3] representing our system can also be visualized graphically as in Figure 2.1. Here, circles denote states, and events are represented by arcs. The first symbol in each arc label denotes the event, while the symbol following "/" denotes the corresponding output. Finally, we mark the controllable events by ":u" and tracking events by "!". Thus, in this example, $X = \{0, 1, 2, 3\}$, $\Sigma = \{\alpha, \beta_1, \beta_2, \delta\}$, $\Phi = \{\beta_1, \beta_2\}$, $\Gamma = \{\delta, \beta_1, \beta_2\}$, and $\Xi = \{\alpha, \beta_1, \beta_2\}$.

There are several basic notions that we will need in our investigation. The first is the notion of liveness. Intuitively, a state is alive if it cannot reach any state at which no event is possible. That is, $x \in X$ is alive if $\forall y \in R(A, x)$, $d(y) \neq \emptyset$. Also, we say that $Q \subset X$ is alive if all $x \in Q$, are alive, and we say that A is alive if X is alive.

[3] On occasion, we will construct auxiliary automata for which we will not be concerned with either control or tracking. In such cases we will omit e and t from the specification.

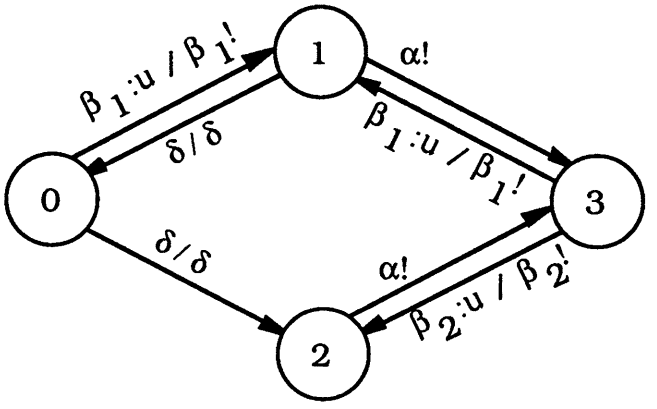

Figure 2.1: A Simple Example

We will assume that this is the case. A second notion that we need is the composition of two automata, $A_i = (G_i, f_i, d_i, h_i)$ which share some common events. Specifically, let $S = \Sigma_1 \cap \Sigma_2$ and, for simplicity, assume that $\Gamma_1 \cap S = \Gamma_2 \cap S$ (i.e., any shared event observable in one system is also observable in the other), $\Phi_1 \cap S = \Phi_2 \cap S$, and $\Xi_1 \cap S = \Xi_2 \cap S$. The dynamics of the composition are specified by allowing each automaton to operate as it would in isolation except that when a shared event occurs, it must occur in both systems. Mathematically, we denote the composition by $A_{12} = A_1 \parallel A_2 = (G_{12}, f_{12}, d_{12}, h_{12}, t_{12})$, where

$$G_{12} = (X_1 \times X_2, \Sigma_1 \cup \Sigma_2, \Phi_1 \cup \Phi_2, \Gamma_1 \cup \Gamma_2, \Xi_1 \cup \Xi_2) \tag{2.6}$$

$$f_{12}(x, \sigma) = f_1(x_1, \sigma) \times f_2(x_2, \sigma) \tag{2.7}$$

$$d_{12}(x) = (d_1(x_1) \cap \overline{S}) \cup (d_2(x_2) \cap \overline{S}) \cup (d_1(x_1) \cap d_2(x_2)) \tag{2.8}$$

$$h_{12}(\sigma) = \begin{cases} h_1(\sigma) & \text{if } \sigma \in \Gamma_1 \\ h_2(\sigma) & \text{if } \sigma \in \Gamma_2 \\ \epsilon & \text{otherwise} \end{cases} \tag{2.9}$$

$$t_{12}(\sigma) = \begin{cases} t_1(\sigma) & \text{if } \sigma \in \Xi_1 \\ t_2(\sigma) & \text{if } \sigma \in \Xi_2 \\ \epsilon & \text{otherwise} \end{cases} \tag{2.10}$$

Here we have extended each f_i to all of $\Sigma_1 \cup \Sigma_2$ in the trivial way, namely, $f_i(x_i, \sigma) = x_i$ if $\sigma \notin \Sigma_i$. Note also that h_{12} and t_{12} are well-defined.

2.2 Languages

Let L be a regular language over an alphabet Σ (see [8]). As in [8], let (A_L, x_0) be a minimal recognizer for L. Given a string $s \in L$, if $s = pqr$ for some p, q and r over Σ then we say that p is a prefix of s and r is a suffix of s. We also use s/pq to denote the suffix r. Also, we say that q is a substring of s. Finally, we need the following characterization of the notion of liveness in the context of languages:

Definition 2.1 Given L, $s \in L$ has an infinite extension in L if for all integers $i \geq |s|$, there exists $r \in L$, $|r| = i$ such that s is a prefix of r. L is prefix closed if all the prefixes of any $s \in L$ are also in L. L is a complete language if each string in L has an infinite extension in L and L is prefix closed. ❑

For any language L, we let L^c denote the prefix closure of L, i.e.

$$L^c = \{p \in \Sigma^* | p \text{ is a prefix of some } s \in L\} \tag{2.11}$$

2.3 Forced Events

In our development we will find it necessary to construct automata in which certain events can be forced to occur regardless of the other events defined at the current state and in fact can only occur if they are forced. The following shows that we can

capture forced events in our present context with a simple construction. Given $x \in X$ let $d_1(x)$ denote the set of forced events defined at x and let $d_2(x)$ denote the other events (controllable or uncontrollable) defined at x. We introduce a new controllable event μ and a new state x' as follows: We redefine $d(x)$ as $d_1(x) \cup \mu$ so that all events defined at x are now controllable and we define $f(x, \mu) = \{x'\}$. Also, we define $d(x') = d_2(x)$ so that $f(x', \sigma) = f(x, \sigma)$ for all $\sigma \in d_2(x)$. If in addition, we impose the restriction that only one event can be enabled at a time at state x, then we can treat forced events as controllable events in our framework. Thus, if we decide to force an event at x, then we enable only that event, and if we decide not to force any events, then we enable μ.

2.4 Stability and Stabilizability

In [6], we define a notion of stability which requires that trajectories go through a given set E infinitely often:

Definition 2.2 Let E be a specified subset of X. A state $x \in X$ is E-pre-stable if there exists some integer i such that every trajectory starting from x passes through E in at most i transitions. The state $x \in X$ is E-stable if A is alive and every state reachable from x is E-pre-stable. The DEDS is E-stable if every $x \in X$ is E-stable (Note that E-stability for all of A is identical to E-pre-stability for all of A). ❑

By a cycle, we mean a finite sequence of states $x_1, x_2, \ldots x_k$, with $x_k = x_1$, so that there exists an event sequence s that permits the system to follow this sequence of states. In [6] we show that E-stability is equivalent to the absence of cycles that do not pass through E. We refer the reader to [6] for a more complete discussion of this subject and for an $O(n^2)$ test for E-stability of a DEDS. Finally, we note that in [6]

and Definition 2.2 we require livenes in order for a system to be stable. However, on occasion, in this paper (see, for example, the following section on the tracking alphabet), it is useful to allow trajectories to die provided that they die in E. It is straightforward to check that all of our results in [6] also hold for this slightly more general notion of stability.

In [6], we also study stabilization by state feedback. Here, a state feedback law is a map $K : X \to U$ and the resulting closed-loop system is $A_K = (G, f, d_K, h, t)$ where

$$d_K(x) = (d(x) \cap K(x)) \cup (d(x) \cap \overline{\Phi}) \tag{2.12}$$

Definition 2.3 A state $x \in X$ is E-pre-stabilizable (respectively, E-stabilizable) if there exists a state feedback K such that x is E-pre-stable (respectively, E-stable) in A_K. The DEDS is E-stabilizable if every $x \in X$ is E-stabilizable. ❑

If A is E-stabilizable, then (as we show in [6]), there exists a state feedback K such that A_K is E-stable. We refer the reader to [6] for a more complete discussion of this subject and for an $O(n^3)$ test for E-stabilizability of a DEDS, which also provides a construction for a stabilizing feedback.

2.5 Tracking Alphabet

The tracking alphabet Ξ provides the flexibility to specify strings that we desire to track over an alphabet which may be much smaller than the entire event alphabet Σ. Note that if there exists a cycle in A that consists solely of events that are not in Ξ, then the system may stay in this cycle indefinitely, generating no event in Ξ. To avoid this possibility, we assume that it is not possible for our DEDS to generate arbitrarily long sequences of events in $\overline{\Xi}$. A necessary and sufficient condition for

checking this is that if we remove the events in Ξ, the resulting automaton $A|\overline{\Xi}$ must be D_t-stable, where

$$D_t = \{x \in X | d(x) \subset \Xi\} \tag{2.13}$$

This is not difficult to check and will be assumed.

2.6 Invariance

In [6], we define the following notion of dynamic invariace in order to characterize stability in terms of pre-stability:

Definition 2.4 A subset Q of X is f-invariant if $f(Q,d) \subset Q$ where

$$f(Q,d) = \bigcup_{x \in Q} f(x, d(x))$$

. ❑

In [6] we show that the maximal stable set is the maximal f-invariant set in the maximal pre-stable set.

In the context of control, the following notion, also presented in [6], is a well-known extension of f-invariance:

Definition 2.5 A subset Q of X is (f,u)-invariant if there exists a state feedback K such that Q is f-invariant in A_K. ❑

However, recall that in general we also need to preserve liveness. Thus we have the following:

Definition 2.6 A subset Q of X is a sustainably (f,u)-invariant set if there exists a state feedback K such that Q is alive and f-invariant in A_K. ❑

Given any set $V \subset X$, there is a maximal sustainably (f,u)-invariant subset W of V with a corresponding unique minimally restrictive feedback K. That is K disables as few events as possible in order to keep the state within W.

2.7 Observability and Observers

In [3], we term a system observable if the current state is known perfectly at intermittent but not necessarily fixed intervals of time. Obviously, a necessary condition for observability is that it is not possible for our DEDS to generate arbitrarily long sequences of unobservable events, i.e., events in $\overline{\Gamma}$, the complement of Γ. A necessary and sufficient condition for checking this is that if we remove the observable events, the resulting automaton $A|\overline{\Gamma} = (G, f, d \cap \overline{\Gamma}, h, t)$ must be D_O-stable, where D_O is the set of states that only have observable transitions defined, i.e., $D_O = \{x \in X | d(x) \cap \overline{\Gamma} = \emptyset\}$. This is not difficult to check and will be assumed.

Let us now introduce some notation that we will find useful:

- Let $x \rightarrow^s y$ denote the statement that state y is reached from x via the occurence of event sequence s. Also, let $x \rightarrow^* y$ denote that x reaches y in any number of transitions, including none. For any set $Q \subset X$ we define the reach of Q in A as:

$$R(A, Q) = \{y \in X | \exists x \in Q \text{ such that } x \rightarrow^* y\} \tag{2.14}$$

- Let

$$Y_0 = \{x \in X | \nexists y \in X, \sigma \in \Sigma, \text{ such that } x \in f(y, \gamma)\} \tag{2.15}$$

$$Y_1 = \{x \in X | \exists y \in X, \gamma \in \Gamma, \text{ such that } x \in f(y, \gamma)\} \tag{2.16}$$

$$Y = Y_0 \cup Y_1 \tag{2.17}$$

Thus, Y is the set of states x such that either there exists an observable transition defined from some state y to x (as captured in Y_1) or x has no transitions defined to it (as captured in Y_0). Let $q = |Y|$.

- Let $L(A, x)$ denote the language generated by A, from the state $x \in X$, i.e., $L(A, x)$ is the set of all possible event trajectories of finite length that can be generated if the system is started from the state x. Also, let $L(A) = \bigcup_{x \in X} L(A, x)$ be the set of all event trajectories that can be generated by A.

In [3], we present a straightforward design of an observer that produces "estimates" of the state of the system after each observation $\gamma[k] \in \Gamma$. Each such estimate is a subset of Y corresponding to the set of possible states into which A transitioned when the last observable event occurred. Mathematically, if we let a function $\hat{\mathbf{x}} : h(L(A)) \rightarrow 2^Y$ denote the estimate of the current state given the observed output string $t \in h(L(A))$, then

$$\hat{\mathbf{x}}(t) = \{x \in Y | \exists y \in X \text{ and } s \in L_f(A, y) \text{ such that } h(s) = t \text{ and } x \in f(y, s)\} \quad (2.18)$$

The observeris a DEDS which realizes this function. Its state space is a subset Z of 2^Y, and its full set of events and set of observable events are both Γ. Suppose that the present observer estimate is $\hat{x}[k] \in Z$ and that the next observed event is $\gamma[k+1]$. The observer must then account for the possible occurence of one or more unobservable events prior to $\gamma[k+1]$ and then the occurrence of $\gamma[k+1]$:

$$\hat{x}[k+1] = w(\hat{x}[k], \gamma[k+1]) \triangleq \bigcup_{x \in R(A|\Gamma, \hat{x}[k])} f(x, \gamma[k+1]) \quad (2.19)$$

$$\gamma[k+1] \in v(\hat{x}[k]) \triangleq h(\bigcup_{x \in R(A|\Gamma, \hat{x}[k])} d(x)) \quad (2.20)$$

The set Z is then in the reach of $\{Y\}$ using these dynamics, i.e., we start the observer in the state corresponding to a complete lack of state knowledge and let it evolve.

Our observer then is the DEDS $O = (F, w, v, i)$, where $F = (Z, \Gamma, \Phi, \Gamma)$ and i is the identity output function. In some cases, we will treat the observer as a controlled system and discuss stabilizing it. Then, Equation 2.20 becomes

$$\gamma[k+1] \in v(\hat{x}[k]) \triangleq h(\bigcup_{x \in R(A|\overline{\Gamma}, \hat{x}[k])}(d(x) \cap u[k]) \cup (d(x) \cap \overline{\Phi})) \tag{2.21}$$

In [3], we show that a system A is observable iff O is stable with respect to its singleton states. We also show that if A is observable then all trajectories from an observer state pass through a singleton state in at most q^2 transitions.

In [3] we also define a notion of recurrency. In particular, we say that a state x is a recurrent state if it can be reached by an arbitrarily long string of events. We let Z_r denote the set of recurrent states of the observer O.

2.8 Compensators

In [5], we define a compensator as a map $C : X \times \Sigma^* \rightarrow U$ which specifies the set of controllable events that are enabled given the current state and the entire event trajectory up to present time. Given a compensator C, the closed loop system A_C is the same as A but with

$$\sigma[k+1] \in d_C(x[k], s[k]) \triangleq (d(x[k]) \cap C(x[k], s[k])) \cup (d(x) \cap \overline{\Phi}) \tag{2.22}$$

where $s[k] = \sigma[0] \cdots \sigma[k]$ with $\sigma[0] = \epsilon$. Here we have somewhat modified notation in that we allow d_C to depend both on $x[k]$ and $s[k]$. It is not difficult to show that we can always write A_C as an automaton (with corresponding "d" depending only on the state), which will take values in an expanded state space, representing the cross-product of the state spaces of A and C. For an arbitrary choice of C, its state space (i.e., an automaton realizing the desired map) may be infinite. As show in [5]

for our purposes we can restrict attention to compensators which can be realized by finite state machines.

In [4] we define an output compensator as a map $C : \Gamma^* \to U$. Then, the closed loop system A_C is the same as A but with:

$$\sigma[k+1] \in d_C(x[k], s[k]) \triangleq (d(x[k]) \cap C(h(s[k]))) \cup (d(x) \cap \overline{\Phi}) \tag{2.23}$$

One constraint we wish to place on our compensators is that they preserve liveness. Thus, suppose that we have observed the output string s, so that our observer is in $\hat{\mathbf{x}}(s)$ and our control input is $C(s)$. Then, we must make sure that any x reachable from any element of $\hat{\mathbf{x}}(s)$ by unobservable events only is alive under the control input $C(s)$. That is, for all $x \in R(A|\overline{\Gamma}, \hat{\mathbf{x}}(s))$, $d_C(x, s)$ should not be empty. This leads to the following:

Definition 2.7 Given $Q \subset X$, $F \subset \Phi$, F is Q-compatible if for all $x \in R(A|\overline{\Gamma}, Q)$, $(d(x) \cap F) \cup (d(x) \cap \overline{\Phi}) \neq \emptyset$. An observer feedback $K : Z \to U$ is A-compatible if for all $\hat{x} \in Z$ $K(\hat{x})$ is $\hat{x}$-compatible. A compensator $C : \Gamma^* \to U$ is A-compatible if for all $s \in h(L(A))$, $C(s)$ is $\hat{\mathbf{x}}(s)$-compatible. ❑

2.9 Stabilization by Output Feedback

In [4] we define a notion of stabilization by output feedback which requires that we can force the trajectories to go through E infinitely often using an output compensator:

Definition 2.8 A is output stabilizable (respectively, output pre-sta-bilizable) with respect to E if there exists an output compensator C such that A_C is E-stable (respectively, E-pre-stable). We term such a compensator an output stabilizing (respectively, output pre-stabilizing) compensator. ❑

Note that this definition implicitly assumes that there exists an integer n_s such that the trajectories in A_C go through E in at most n_s observable transitions. In [4] we show that n_s is at most q^3, and, using this bound, we show that output pre-stabilizability and liveness are necessary and sufficient for output stabilizability, as is the case for stabilizability and pre-stabilizability.

2.10 Eventual Restrictability

In [5] we define a notion of restrictability which requires that we can force the system to generate strings in a desired language defined over Ξ:

Definition 2.9 Given $x \in X$ and a complete language L over Ξ, x is L-restrictable if there exists a compensator $C : X \times \Sigma^* \rightarrow U$ such that the closed loop system A_C is alive and $t(L(A_C, x)) \subset L$. Given $Q \subset X$, Q is L-restrictable if all $x \in Q$ are L-restrictable. Finally, A is L-restrictable if X is L-restrictable. ❑

We also define a notion of eventual restrictability which requires that we can restrict the system behavior in a finite number of transitions. In the following, $(\Xi \cup \{\epsilon\})^{n_a}$ denotes the set of strings over Ξ that have length at most n_a:

Definition 2.10 Given $x \in X$ and a complete language L over Ξ, x is eventually L-restrictable if there exists an integer n_a and a compensator $C : X \times \Sigma^* \rightarrow U$ such that the closed loop system A_C is alive and $t(L(A_C, x)) \subset (\Xi \cup \{\epsilon\})^{n_a} L$. Given $Q \subset X$, Q is eventually L-restrictable if all $x \in Q$ are eventually L-restrictable. Finally, A is eventually L-restrictable if X is eventually L-restrictable. ❑

We refer the reader to [5] for a more complete discussion of this subject. We now turn our attention to eventual restrictability using an output compensator. The following,

although not included in [5], is a straightforward use of tools that we have developed in [4] and [5]:

Definition 2.11 Given a complete language L over Ξ we say that A is eventually L-restrictable by output feedback if there exists an integer n_o and an output compensator $C : \Gamma^* \to U$ such that A_C is alive and for all $x \in X$, $t(L(A_C, x)) \subset (\Xi \cup \{\epsilon\})^{n_o} L$. Such a C is called an L-restrictability compensator. ❑

We construct a test for eventual restrictability by output feedback as follows: Given L, let (A_L, x_0^L) be a minimal recognizer for L and let Z_L denote its state space. Let A'_L be an automaton which is the same as A_L except that its state space is $Z'_L = Z_L \cup \{b\}$ where b is a state used to signify that the event trajectory is no longer in L. This is the state we wish to avoid. Also, we let $d'_L(x) = \Xi$ for all $x \in Z'_L$, and

$$f'_L(x,\sigma) = \begin{cases} f_L(x,\sigma) & \text{if } x \neq b \text{ and } \sigma \in d_L(x) \\ \{b\} & \text{otherwise} \end{cases} \tag{2.24}$$

Let O denote the observer for A, let $A(L) = A \parallel A'_L$, and let $O(L) = (G(L), w_L, v_L)$ denote the observer for $A(L)$; however, in this case, since we know that we will start A'_L in x_0^L, we take the state space of $O(L)$ as

$$Z(L) = R(O(L), \{\{x_0^L\} \times \hat{x} | \hat{x} \in Z\}) \tag{2.25}$$

Let

$$V_o = \{\hat{z} \in Z(L) | \text{ for all } (x_L, x_A) \in \hat{z}, x_L \neq b\} \tag{2.26}$$

Let $E(L)$ be the largest subset of V_o which is sustainably (f,u)-invariant in $O(L)$ and for which the associated unique minimally restrictive feedback K^{EL} has the property that for any $\hat{z} \in Z(L)$, $K^{EL}(\hat{z})$ is $\hat{x}(\hat{z})$-compatible where

$$\hat{x}(\hat{z}) = \{x \in X | \exists x_L \in Z_L \text{ such that } (x_L, x) \in \hat{z}\} \tag{2.27}$$

The construction of $E(L)$ and K^{EL} is a slight variation of the algorithm in [6] for the construction of maximal sustainably (f,u)-invariant subsets. Specifically, we begin with any state $\hat{z} \in V_o$. If there are any uncontrollable events taking $\hat{z}$ outside V_o, we delete $\hat{z}$ and work with $V_1 = V_o \setminus \{\hat{z}\}$. If not, we disable only those controllable events which take $\hat{z}$ outside V_o. If the remaining set of events defined at $\hat{z}$ is not $\hat{x}(\hat{z})$-compatible, we delete $\hat{z}$ and work with $V_1 = V_o \setminus \{\hat{z}\}$. If not, we tentatively keep $\hat{z}$ and choose another element of V_o. In this way, we continue to cycle through the remaining elements of V_o. The algorithm converges in a finite number of steps (at most $|V_o|^2|$) to yield $E(L)$ and K^{EL} defined on $E(L)$. For $\hat{z} \in \overline{E(L)}$, we take $K^{EL}(\hat{z}) = \Sigma$ so that no events are disabled.

Consider next the following subset of $E(L)$:

$$E_o(L) = \{\hat{x} \in Z | x_0^L \times \hat{x} \in E(L)\} \tag{2.28}$$

Then,

Proposition 2.12 Given a complete language L over Ξ, A is eventually L-restrictable by output feedback iff there exists an A-compatible state feedback $K : Z \to U$ such that the closed loop system O_K is $E_o(L)$-pre-stable.

Proof: ($\rightarrow$) Straightforward by assuming the contrary.

($\leftarrow$) Let us prove this by constructing the desired compensator $C : \Gamma^* \to U$: Given an observation sequence s, we trace it in O starting from the initial state $\{Y\}$. Let $\hat{x}$ be the current state of O given s. There are two possibilities:

1. Suppose that the trajectory has not yet entered $E_o(L)$. Then we use O and the $E_0(L)$-pre-stabilizing feedback K to compute $C(s)$. In particular,

$$C(s) = (v(\hat{x}) \cap K(\hat{x})) \cup (v(\hat{x}) \cap \overline{\Phi})$$

2. When the trajectory in O enters $E_o(L)$, we switch to using the expanded observer $O(L)$ and $K^{E(L)}$. In particular, let $\hat{x}'$ be the state the trajectory in O enters when it enters $E_o(L)$ for the first time, and let s' be that prefix of s which takes $\{Y\}$ to $\hat{x}'$ in O. Then, we start O_L at the state $x_0^L \times \hat{x}' \in E(L)$, and let it evolve. Suppose that s/s' takes $x_0^L \times \hat{x}'$ to $\hat{z}$ in $O(L)$, then

$$C(s) = (v_L(\hat{z}) \cap K^{E(L)}(\hat{z})) \cup (v_L(\hat{z}) \cap \overline{\Phi})$$

Since this feedback keeps the trajectory of O in $E(L)$ and $E(L) \subset V_o$, the behavior of A is restricted as desired. ❑

Note that since $E(L)$ is the maximal sustainably (f,u)-invariant subset of V_o and $K^{E(L)}$ is unique, the possible behavior of an L-restrictable state x in the closed loop system constructed in the proof is the maximal subset of L to which the behavior of x can be restricted. Note also that if $E_o = \emptyset$, then O cannot be $E_o(L)$-pre-stabilizable and thus A cannot be eventually L-restrictable by output feedback. Finally, if A is eventually L-restrictable by output feedback, then the number of observable transitions until the trajectory is restricted to L is at most n_s.

3 Characterizing Higher-Level Models

In this section, we present a notion of higher-level modelling of DEDS based on a given set of primitives, each of which consists of a finite set of tracking event strings. The idea here is that the occurrence of any string in this set corresponds to some macroscopic event, such as completion of a task, and it is only these macro-events that we wish to model at the higher level. Our modelling concept therefore must address the issues of controlling a DEDS such that its behavior is restricted to these primitives and of being able to observe the occurrences of each primitive. In this section we describe precisely what it means for one DEDS to serve as a higher-level model of another. In subsequent sections we explicitly consider the notion of tasks and the problems of controlling and observing them and the related concept of procedures, defined in terms of sequences of tasks, which allows us to describe higher-level models of interconnections of DEDS, each of which can perform its own set of tasks.

To illustrate our notion of modelling, consider the system in Figure 2.1 and suppose that we wish to restrict its behavior to $(\alpha\beta_1)^*$ by output feedback. In order to do this, we first specify $L_1 = \alpha\beta_1$ as a primitive. For this example it is possible, essentially by inspection, to construct an L_1^{*c}-restrictability compensator $C : \Gamma^* \to U$ that is simpler than the one given in the proof of Proposition 2.12. Specifically, we can take C to be a function of the state of an automaton, illustrated in Figure 3.1, constructed from the observer for the original system simply by deleting the events which the compensator disables. The initial state of this system, as in the observer, is (0,1,2), and for any string s the compensator value $C(s)$ is a function of the state of the modified observer. For example, if the first observed event is δ, the state of the system in Figure 3.1 is (1,2). In the original DEDS of Figure 2.1 the event β_1 would

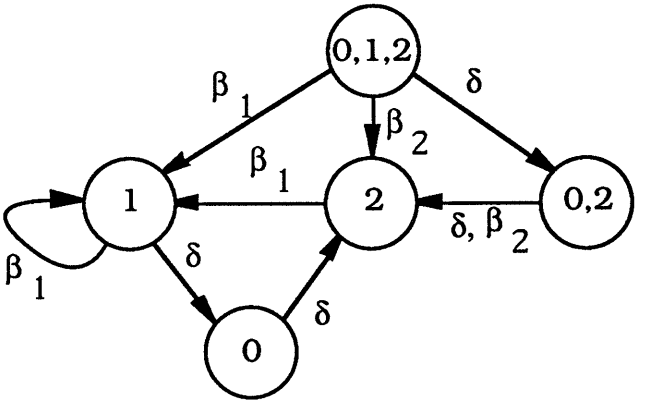

Figure 3.1: Illustrating the Compensator for Eventual L_1^{*c}-Restrictability by Output Feedback

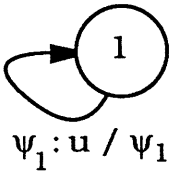

Figure 3.2: Model of Task 1

be possible from state 0; however, as illustrated in Figure 3.1, we disable β_1 so that the next observable event will either be δ (from 0) or β_2 (from 3 after the occurrence of α from 2). It is not difficult to check that the closed loop system A_C is eventually L_1^{*c}-restricted, and thus the language eventually generated by A_C can be modelled at a higher level by the automaton in Figure 3.2 for which ψ_1 represents an occurrence of $\alpha\beta_1$.

In order for this automaton, with ψ_1 observable, to truly model A_C, it must be true that we can in fact detect every occurrence of $\alpha\beta_1$ in A_C, perhaps with some initial uncertainty, given the string of observations of A_C. For example, by inspection of

Figure 3.1, if we observe β_1, we cannot say if $\alpha\beta_1$ has occured or not, but if we observe $\beta_1\beta_1$, we know that $\alpha\beta_1$ must have occured at least once. Likewise, $\beta_2\beta_1$ corresponds to one occurrence of $\alpha\beta_1$, $\delta\beta_2\beta_1\delta\delta\beta_1$ corresponds to two occurrences of $\alpha\beta_1$, etc. In general, after perhaps the first occurrence of β_1, every occurrence of β_1 corresponds to an occurrence of $\alpha\beta_1$ and therefore, we can detect occurrences of ψ_1 from observations in A_C. The definition we give in this section will then allow us to conclude that the automaton in Figure 3.2 models the closed loop system A_C.

To begin our precise specification of higher-level models, let us first introduce a function that defines the set of strings that corresponds to a primitive: Given alphabets Σ' and Ξ, and a map $H_e : \Sigma' \to 2^{\Xi^*}$, if for all $\sigma \in \Sigma'$ $H_e(\sigma)$ is a collection of finite length strings, then we term H_e a primitive map. Here $\sigma \in \Sigma'$ is the macroscopic event corresponding to the set of tracking strings $H(\sigma)$ in the original model. We allow the possibility that several strings may correspond to one macroscopic primitive to capture the fact that there may be several ways to complete a desired task.

We will require the following property:

Definition 3.1 A primitive map H_e is termed minimal if for all, not necessarily distinct, $\sigma_1, \sigma_2 \in \Sigma'$ and for all $s \in H_e(\sigma_1)$, no proper suffix of s is in $H_e(\sigma_2)$. ❑

Given a primitive map, we extend it to act on strings over Σ' as follows: We let $H_e(\epsilon) = \epsilon$ and $H_e(s\sigma) = H_e(s)H_e(\sigma)$, where s is a string over Σ' and σ is an element of Σ'. Also, $H_e(s)H_e(\sigma)$ is the set of strings consisting of all possible concatenations of a string in $H_e(s)$ followed by a string in $H_e(\sigma)$. We use the same symbol to denote H_e and its extension to $(\Sigma')^*$.

Proposition 3.2 If H_e is minimal then for all distinct r_1, r_2 such that $r_1, r_2 \neq \epsilon$, $|r_1| \leq |r_2|$, and r_1 is not a suffix of r_2, $\Xi^* H_e(r_1) \cap \Xi^* H_e(r_2) = \emptyset$.

Proof: Assume the contrary, and let $s \in \Xi^* H_e(r_1) \cap \Xi^* H_e(r_2)$. Also let σ_1 (respectively, σ_2) be the last event in r_1 (respectively, r_2). There are two cases here: First, suppose that $\sigma_1 \neq \sigma_2$. Then, there exist distinct $p_1 \in H_e(\sigma_1)$ and $p_2 \in H_e(\sigma_2)$ such that both p_1 and p_2 are suffixes of s. Assume, without loss of generality, that $|p_1| \leq |p_2|$, then p_1 is also a suffix of p_2. But then, H_e cannot be minimal. Now, suppose that $\sigma_1 = \sigma_2$. Thanks to minimality, among all elements of $H_e(\sigma_1)$, only one string, say p can be a suffix of s. Let s' be that prefix of s such that $p = s/s'$. Then, repeat the previous steps using s', and all but the last elements of r_1 and r_2. Since r_1 and r_2 are distinct, and r_1 is not a suffix of r_2, σ_1 will be different from σ_2 at some step and then we will establish a contradiction. Therefore, $\Xi^* H_e(r_1) \cap \Xi^* H_e(r_2) = \emptyset$. ❑

The following result states that we can concatenate minimal primitive maps while preserving minimality:

Proposition 3.3 Given minimal $H_1 : \Sigma_2 \rightarrow 2^{\Sigma_1^*}$ and $H_2 : \Sigma_3 \rightarrow 2^{\Sigma_2^*}$, if we define $H_3 : \Sigma_3 \rightarrow 2^{\Sigma_1^*}$ so that $H_3(\sigma) = H_1(H_2(\sigma))$ for all $\sigma \in \Sigma_3$, then H_3 is a minimal primitive map. Here, since $H_2(\sigma)$ is a set of strings, $H_1(H_2(\sigma))$ is the set of strings resulting from applying H_1 to each string in $H_2(\sigma)$.

Proof: Assume contrary, then there exists, $\sigma_1, \sigma_2 \in \Sigma_3$, $s \in H_3(\sigma_1)$, and a suffix r of s so that $r \in H_3(\sigma_2)$. Let $s' \in H_2(\sigma_1)$ and $r' \in H_2(\sigma_2)$ such that $s \in H_1(s')$ and $r \in H_1(r')$. Then, by minimality of H_2, r' cannot be a suffix of s' and s' cannot be a suffix of r' either. Also, since r is a suffix of s, $s \in \Sigma_1^* H_1(r')$. Then, thanks to Proposition 3.2, H_1 cannot be minimal, and we establish a contradiction. Therefore, H_3 must be minimal. ❑

Now, let us proceed with defining our notion of modelling. In particular, given two automata $A = (G, f, d, h, t)$ and $A' = (G', f', d', h', t')$, we wish to specify when

A' is an H_e-model of the system A, where $H_e : \Sigma' \rightarrow \Xi$ is a minimal primitive map. Two important properties that we will require of our models are the following:

1. **Restrictability:** We will require that if we can restrict the behavior of the macroscopic model to some complete language $L \subset \Sigma'^*$, then we can also restrict the original system to $H_e(L)^c$. Note that we have defined L over Σ', instead of Ξ'. Roughly speaking, we have done this since if all languages of interest are over the higher-level tracking alphabet Ξ', then we can perhaps choose a simpler macroscopic model completely over Ξ'. However, the alphabet Ξ' will still be useful in defining different levels of modelling (see Proposition 3.6).

2. **Detectability** We will also require that for any lower-level string s in $L(A)$ such that $t(s)$ is in $H_e(p)$ for some string p in the macroscopic system,

   (a) we can reconstruct p, after some delay, using the lower-level observation $h(s)$ of s, and

   (b) for any string r so that s is a suffix of r, the reconstruction acting on $h(r)$ results in a string that ends with the reconstruction of $h(s)$.

   Note that minimality implies the following: If we let $H_e^{-1}(t(s))$ denote the set of strings $p \in \Sigma'^*$ such that $t(s) \in H_e(p)$, then, thanks to minimality, $H_e^{-1}(t(s))$ is single valued. Thus, in order to satisfy the first condition of detectability, we need to be able to reconstruct $H_e^{-1}(t(s))$ from $h(s)$. The second condition deals with the issue of start-up. Specifically, in our framework of eventual restrictability, we allow for the possibility of a transient start-up period in which the lower-level may generate a short tracking event sequence that does not correspond to any primitive. What (b) requires is that the reconstruction can

recognize and "reject" such finite length start-up strings.

Definition 3.4 Given two DEDS A and A', and a minimal primitive map $H_e : \Sigma' \rightarrow 2^{\Xi^*}$, we say that A' is an $\underline{H_e\text{-model}}$ of A if there exists a map $H_o : \Gamma^* \rightarrow \Sigma'^*$ and an integer n_d such that[4]:

1. **Restrictability:** For all complete $L \subset \Sigma'^*$ such that A' is eventually L-restrictable, A is eventually $H_e(L)^c$-restrictable by output feedback.
2. **Detectability:** For all $s \in L(A)$, such that $t(s) \in H_e(p)$ for some $p \in L(A')$,
   (a) $p \in (\Sigma' \cup \{\epsilon\})^{n_d} H_o(h(s))$, and
   (b) for all $r \in \Sigma^* s$, $H_o(h(r)) \in \Sigma'^* H_o(h(s))$.

□

Note that this concept of modelling provides a method of both spatial and temporal aggregation, as we will see, since A' may frequently be constructed to have many fewer states than A and sets of strings in A can be represented by a single event in A'. For example, all states in Figure 2.1 are represented by a single state in Figure 3.2, and $\alpha\beta_1$ is represented by ψ_1.

The following result, which immediately follows from Definition 3.4, states that the concept of modelling is invariant under compensation:

[4]We have chosen in our definition to look at the larger class of macroscopic languages to which A is eventually restrictable by full state feedback, rather than only with output feedback. All of our results carry over if we use this weaker notion of restrictability at the higher level. Similarly in our definition of detectability we have required the stronger condition that from lower level observations, we can reconstruct the entire upper-level event trajectory, not just the part in Γ'. Again, we can carry all of our development over to the weaker case. As we will see, this stronger definition suffices for our purposes

Proposition 3.5 If A' is an H_e-model of A then for any compensator $C' : \Gamma'^* \rightarrow U'$ for A', there exists a compensator $C : \Gamma^* \rightarrow U$ for A such that $A'_{C'}$ is an H_e-model of A_C with the same H_o. ❑

In general, we may be interested in several different levels of aggregation. Thus, we need the following result which states that a higher-level model also models automata at all lower levels:

Proposition 3.6 Given the automata $A = (G, f, d, h, t)$, $A' = (G', f', d', h', t')$, and $A'' = (G'', f'', d'', h'', t'')$, and minimal primitive maps $H'_e : \Sigma' \rightarrow 2^{\Xi^*}$ and $H''_e : \Sigma'' \rightarrow 2^{\Xi'^*}$, so that A' is an H'_e-model of A with H'_o and A'' is an H''_e-model of A' with H''_o, define $\pi : \Xi' \rightarrow 2^{\Sigma'^*}$ so that $\pi(\sigma) = \sigma(\overline{\Xi'} \cup \{\epsilon\})^{|X'|}$ for $\sigma \in \Xi'$ and define $H_e : \Sigma'' \rightarrow 2^{\Xi^*}$ as $H_e(\sigma) = H'_e(\pi(H''_e(\sigma)))$ for $\sigma \in \Sigma''$. Then,

1. H_e is a minimal primitive map.
2. A'' is an H_e-model of A with $H_o(s) = H''_o(h'(H'_o(s)))$ for all $s \in \Gamma^*$.

Proof: 1. Clearly, π is a minimal primitive map. Then, by Proposition 3.3, H_e is also a minimal primitive map.

2. Restrictability: If A'' is eventually L-restrictable, then A' is eventually $H''_e(L)^c$-restrictable by output feedback $\rightarrow$ A' is eventually $H''_e(L)^c$-restrictable $\rightarrow$ A' is eventually $\pi(H''_e(L)^c)$-restrictable $\rightarrow$ A is eventually $H_e(L)^c$-restrictable by output feedback.

Detectability: Straightforward. ❑

4 Aggregation

In this section, we use the concept of modelling of the previous section to present an approach for the aggregation of DEDS. What we have in mind is the following paradigm. Suppose that our system is capable of performing a set of tasks, each of which is a primitive as defined in the previous section. What we would like to do is to design a compensator that accepts as inputs requests to perform particular tasks and then controls A so that the appropriate task is performed. Assuming that the completion of this task is detected, we can construct a higher level and extremely simple standard model for our controlled system: tasks are requested and completed. Such a model can then be used as a building block for more complex interconnections of task-oriented automata and as the basis for the closed loop following of a desired task schedule.

In the first subsection we define tasks and several critical properties of sets of tasks and their compensators. Roughly speaking we would like tasks to be uniquely identifiable segments of behavior of A that in addition do not happen "by accident" during task set-up. More precisely, we introduce the notion of independence of tasks which states that no task is a subtask of another (so that all tasks describe behavior at roughly the same level of granularity) and the notion of a consistent compensator, which, while setting up to perform a desired task, ensures that no other task is completed. With such a set of tasks and compensators we can be assured that a desired task sequence can be followed, with task completions seperated by at most short set-up periods. In Section 4.2 we discuss the property of task observability, i.e., the ability to detect all occurrences of specified tasks. In Section 4.3 we then put these pieces together to construct a special higher-level model which we refer to as

task standard form.

4.1 Reachable Tasks

Our model of a task is a finite set of finite length strings, where the generation of any string in the set corresponds to the completion of the task. Let T be the index set of a collection of tasks, i.e., for any $i \in \mathsf{T}$ there is a finite set L_i of finite length strings over Ξ that represents task i. In our development we will need a similar but stronger notion than that of minimality used in the previous section. We let $L_T = \cup_{i \in \mathsf{T}} L_i$ and define the following:

Definition 4.1 Given T, we say that T is an independent task set if for all $s \in L_T$, no substring of s, except for itself, is in L_T. ❑

Then when we look at a tracking sequence there is no ambiguity concerning what tasks have been completed and which substring corresponds to which task. Note that if T is an independent set, then the minimal recognizer (A_T, x_0) for all of L_T has a single final state x_f, i.e., all strings in L_T take x_0 to x_f, and x_f has no events defined from it (since L_T is a finite set). Furthermore, for each $i \in \mathsf{T}$, the minimal recognizer $(A_{L_i}, x_0^{L_i})$ also has a single final state $x_f^{L_i}$ which has no events defined from it.

Let us define a second task $L_2 = \alpha\beta_2$ in addition to the task $L_1 = \alpha\beta_1$ for the example in Figure 2.1. Note that this system is in fact eventually L_1^{*c} and L_2^{*c}-restrictable. We term such tasks reachable:

Definition 4.2 A task $i \in \mathsf{T}$ is reachable if A is eventually L_i^{*c}-restrictable. T is a reachable set if each $i \in \mathsf{T}$ is reachable. ❑

Definition 4.3 Task $i \in \mathbf{T}$ is reachable by output feedback if A is eventually L_i^{*c}-restrictable by output feedback. $\mathbf{T}$ is reachable by output feedback if each $i \in \mathbf{T}$ is reachable by output feedback. ❑

Given a task $i \in \mathbf{T}$ that is reachable by output feedback, let $C_i : \Gamma^* \rightarrow U$ be an L_i^{*c}-restrictability compensator. Consider the possible behavior when we implement this compensator. Note that states in $E_o(L_i^{*c})$, as defined in Section 2.10, are guaranteed to generate a sublanguage of L_i^{*c} in the closed loop system. However, for any other state $\hat{x} \in Z$, although we cannot guarantee that L_i^{*c} will be generated given the particular knowledge of the current state of the system (i.e., given that the system is in some state in $\hat{x}$), it may still be possible for such a string to occur. Furthermore, in general, a string in L_j, for some other j, may be generated from a state $x \in \hat{x}$ before the trajectory in O reaches $E_o(L_i^{*c})$. If in fact a string in L_j is generated from some $x \in \hat{x}$, then task j will have been completed while the compensator was trying to set-up the system for task i. Since this is a mismatch between what the compensator is trying to accomplish and what is actually happening in the system, we will require that it cannot happen. We define this property as follows (we state the definition for recurrent observer states, allowing for mismatch for a bounded number of transitions at the overall start-up of the system):

Definition 4.4 Given a reachable task $i \in \mathbf{T}$ and an L_i^{*c}-restrictability compensator C_i, C_i is consistent with $\mathbf{T}$ if for all $\hat{x} \in Z_r \cap \overline{E_o(L_i^{*c})}$, for all $x \in \hat{x}$, and for all $s \in L(A_{C_i}, x)$, $t(s) \notin L_T$. ❑

Now, let us consider testing the existence of and constructing consistent restrictability compensators. Note that we only need to worry about forcing the trajectory in O into $E_o(L_i^{*c})$ without completing any task along the way. Once that is done, restricting

the behavior can be achieved by the compensator defined in the proof of Proposition 2.12. First, we need a mechanism to recognize that a task is completed. Thus, let (A_T, x_0) be a minimal recognizer for L_T with the final state x_f. Let X_T be the state space of A_T. Since not all events are defined at all states in X_T, we do the following: we add a new state, say state g to the state space of A_T, and for each event that is not previously defined at states in X_T we define a transition to state g. Thus if A_T enters state g, we know that the tracking event sequence generated starting from x_0 and ending in g is not the prefix of any task sequence. Also, to keep the automaton alive, we define self-loops for all events in Ξ at states g and x_f. Let A'_T be this new automaton. Given a string s over Ξ, if s takes x_0 to g in A'_T then no prefix of s can be in L_T. If, on the other hand, the string takes x_0 to x_f then some prefix of this string must be in L_T. Now, let $O' = (G', w', v')$ be the observer for $A \parallel A'_T$. We let the state space Z' of O' be the range of initial states

$$Z'_0 = \{\hat{x} \times \{x_0\} | \hat{x} \subset Z_r\} \tag{4.1}$$

i.e., $Z' = R(O', Z_0)$. Let $p : Z' \rightarrow Z_r$ be the projection of Z' into Z_r, i.e., given $\hat{z} \in Z'$, $p(\hat{z}) = \bigcup_{(x_1,x_2)\in\hat{z}}\{x_1\}$. Also, let $E'_o = \{\hat{z} \in Z' | p(\hat{z}) \in E_o(L_i^{*c})\}$. Our goal is to reach E'_o from the initial states Z'_0 while avoiding the completion of any task. Once the trajectory arrives at E'_o further behavior can be restricted as desired. So, we remove all transitions from states in E'_o and instead create self loops in order to preserve liveness. Let $O'' = (G', w'', v'')$ represent the modified automaton. Let us now consider the set of states in which we need to keep the trajectory. These are the states that cannot correspond to a completion of any task. Thus, we need to keep the trajectories in the set

$$E'' = \{\hat{z} \in Z' | \forall (x_1, x_2) \in \hat{z}, x_2 \neq x_f\} \tag{4.2}$$

Let V' be the maximal (f,u)-invariant subset of E', and let $K^{V'}$ be the corresponding A-compatible and minimally restrictive feedback. In order for a consistent compensator to exist, Z_0' must be a subset of V'. Assuming this to be the case, we need to steer the trajectories to E_o' while keeping them in V'. Therefore, we need to find a feedback $K'' : Z' \to U$ so that Z' is E_o'-pre-stable in $O''_{K^{V'}}$ and so that the combined feedback $K : Z' \to U$ defined by

$$K(\hat{z}) = K^{V'}(\hat{z}) \cap K''(\hat{z}) \tag{4.3}$$

for all $\hat{z} \in Z'$ is A-compatible. The construction of such a K, if it exists, proceeds much as in our previous construction in Section 2. Thanks to the uniqueness of $K^{V'}$, if we cannot find such a feedback, then a consistent restrictability compensator cannot exist. In the analysis in this paper, we assume that consistent compensators exist. That is, we assume that for each task $Z_0' \subset V'$ and K exists.

Finally, let us outline how we put the various pieces together to construct a consistent compensator C_i for task i: Given an observation sequence s, we trace it in O starting from the initial state $\{Y\}$. Let $\hat{x}$ be the current state of O given s. There are three possibilities:

1. Suppose that $\hat{x} \notin Z_r$ and the trajectory has not entered $E_o(L_i^{*c})$ yet. Then, we use O and an $E_o(L_i^{*c})$-pre-stabilizing feedback to construct $C_i(s)$ as explained in the proof of Proposition 2.12.

2. Suppose that $\hat{x} \in Z_r$ and the trajectory has not entered $E_o(L_i^{*c})$ yet. Then, we use the observer O'' and the feedback K defined above. In particular, let $\hat{x}'$ be the state in the observer O into which the trajectory moves when it enters Z_r for the first time, and let s' be that prefix of s which takes $\{Y\}$ to $\hat{x}'$ in O. Then,

we start O'' at state $\hat{x}' \times x_o$ and let it evolve. Suppose that s/s' takes $\hat{x}' \times x_0$ to $\hat{z}$ in O'' then

$$C_i(s) = (v''(\hat{z}) \cap K(\hat{z})) \cup (v''(\hat{z}) \cap \overline{\Phi}) \tag{4.4}$$

3. When the trajectory enters $E_o(L_i^{*c})$, we switch to using $O(L_i^{*c})$ and the (f,u)-invariance feedback $K^{L_i^{*c}}$. $C_i(s)$ in this case can be constructed as explained in the proof of Proposition 2.12.

In order to develop a complete higher-level modelling methodology, we need to describe explicitly an overall compensator which responds to requests to perform particular tasks by enabling the appropriate compensator C_i. Given a set of p tasks $\mathbf{T}$, reachable by output feedback, and a task $i \in \mathbf{T}$, let $C_i : \Gamma^* \to U$ denote the compensator corresponding to task i. The compensator C that we construct admits events corresponding to requests for tasks as inputs and, depending on the inputs, C switches in an appropriate fashion between C_i. In order to model this, we use an automaton illustrated in Figure 4.1, which has p states, where state i corresponds to using the compensator C_i to control A. For each i, τ_i^F is a forced event, corresponding to switching to C_i. Let $\Phi_T = \{\tau_1^F, \ldots, \tau_p^F\}$ and $U_T = 2^{\Phi_T}$. The input to C is a subset of Φ_T, representing the set of tasks which are requested at present. The compensator responds to this input as follows: Suppose that C is set-up to perform task i. There are three possibilities: (1) If the input is the empty set, then C disables all events in A, awaiting future task requests; (2) if the input contains τ_i^F, then C will not force any event but continue performing task i (thereby avoiding an unnecessary set-up transient); (3) Finally, if the input is not empty but it does not contain τ_i^F, then C will force one of the events in this set. At this level of modelling, we do not care which event C decides to force. Thus, we define $C : U_T \times \Gamma^* \to U \times \Phi_T$ so that given an input

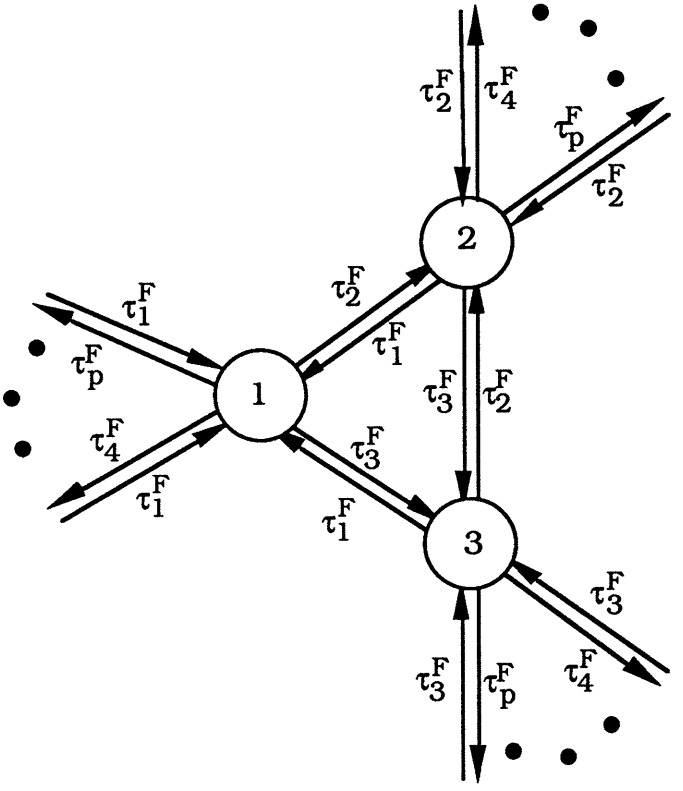

Figure 4.1: An Automaton to Construct C

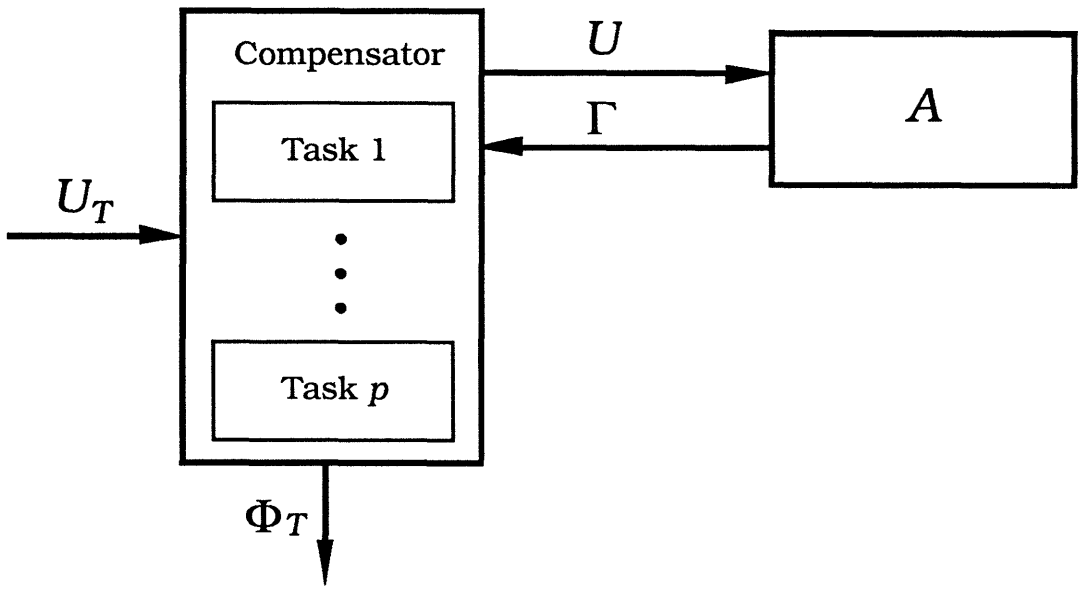

Figure 4.2: Block Diagram for A_C

in U_T, C chooses the appropriate input ($\in U$) to A and generates the forced events ($\in \Phi_T$) as explained above. If the action of C corresponds to a switch from one task to another, the compensator C_i is initialized using the approach described previously. Specifically, suppose that the observer is in state $\hat{x}$ right before τ_i^F is forced. Consider the three cases described previously for C_i: If $\hat{x} \notin Z_r$ and $\hat{x} \notin E_o(L_i^{*c})$, then we use O starting from the initial state $\hat{x}$ and an $E_o(L_i^{*c})$-pre-stabilizing feedback. If $\hat{x} \in Z_r$ and $\hat{x} \notin E_o(L_i^{*c})$, then we start O'' at state $\hat{x} \times x_0$ and use the compensator described previously to drive the system to the desired set of states. Finally, if $\hat{x} \in E_o(L_i^{*c})$, then we start $O(L_i^{*c})$ at state $x_0^{L_i^{*c}} \times \hat{x}$, where $x_0^{L_i^{*c}}$ is the initial state of the minimal recognizer for L_i^{*c}, and we use the (f,u)-invariant feedback $K^{L_i^{*c}}$. A block diagram for A_C is illustrated in Figure 4.2.

4.2 Observable Tasks

In this section, we define a notion of observability for tasks which allows us to detect all occurrences of a given task. Consistent with our definition of detectability, we define task observability after an initial start-up transient. Specifically, we focus on detecting occurrences of tasks from that point in time at which the observer enters a recurrent state. One could, of course, consider the stricter condition of observability without any knowledge of the initial state, but this would seem to be a rather strong condition. Rather, our definition can be viewed either as allowing a short start-up period or as specifying the level of initial state knowledge required in order for task detection to begin immediately (i.e., we need our initial state uncertainty to be confined to an element of Z_r).

Definition 4.5 A task $i \in \mathbf{T}$ is observable if there exists a function $\mathcal{I} : Z_r \times L(O, Z_r) \rightarrow \{\epsilon, \psi_i^F\}$ so that for all $\hat{x} \in Z_r$ and for all $x \in \hat{x}$, $\mathcal{I}$ satisfies

1. $\mathcal{I}(\hat{x}, h(s)) = \psi_i^F$ for all $s \in L(A, x)$ such that $s = p_1 p_2 p_3$ for some $p_1, p_2, p_3 \in \Sigma^*$ for which $t(p_2) \in L_i$, and
2. $\mathcal{I}(\hat{x}, h(s)) = \epsilon$ for all other $s \in L(A, x)$.

A set of tasks $\mathbf{T}$ is observable if each $i \in \mathbf{T}$ is observable. ❑

Since we assume that tasks are reachable throughout this paper and will use task observability only in conjunction with task control, we will construct a test for the observability of task i assuming that it is reachable and that we are given an L_i^{*c}-restrictability compensator C_i which is consistent with $\mathbf{T}$. Furthermore, thanks to consistency, we only need to construct $\mathcal{I}$ for $\hat{x} \in E_o(L_i^{*c})$ and for strings s such that $t(s) \in L_i^{*c}$. First, we let $A'_{L_i} = (G'_{L_i}, f'_{L_i}, d'_{L_i})$ be the same as the recognizer A_{L_i}

but with a self-loop at the final state $x_f^{L_i}$ for each $\sigma \in \Xi$. Now, let $Q = (G_Q, f_Q, d_Q)$, with state space X_Q, denote the live part of $A'_{L_i} \parallel A$, i.e., X_Q is the set of states x in $X'_{L_i} \times X$ so that there exists an arbitrarily long string in $L(A'_{L_i} \parallel A, x)$. In fact, note that for each $x \in X$ such that $(x_0^{L_i}, x) \in X_Q$, there exists $s \in L(A, x)$ so that $t(s) \in L_i$. Finally, let $O_Q = (F_Q, w_Q, v_Q)$ be the observer for Q so that the state space Z_Q of O_Q is the reach of

$$Z_{Q0} = \bigcup_{\hat{x} \in E_o(L_i^{*c})} (\{x_0^{L_i}\} \times \hat{x}) \cap X_Q \tag{4.5}$$

in O_Q, i.e., $Z_Q = R(O_Q, Z_{Q0})$. Note that if i is observable, then the last event of each string in L_i must be an observable event. Assuming that this is the case, let

$$E_Q = \{\hat{z} \in Z_Q | \exists (x, y) \in \hat{z} \text{ such that } x = x_f^{L_i}\} \tag{4.6}$$

Given the observations on A_{C_i} let us first trace the trajectory in the observer O. At some point in time, O will enter some state $\hat{x} \in E_o(L_i^{*c})$. When this happens we know that the system starts tracking task i. At this point, let us start tracing the future observations in O_Q starting from the state $(\{x_0^{L_i}\} \times \hat{x}) \cap X_Q$. This trajectory will enter some $\hat{z} \in E_Q$ at some point in time. At this point, we know that task i may have been completed. However, for task observability, we need to be certain that task i is completed whenever it is actually, completed. Thus, for an observable task, it must be true that for all $\hat{z} \in E_Q$ and for all $(x, y) \in \hat{z}$, $x = x_f^{L_i}$. In this case we can define $\mathcal{I}$ to be ϵ until the trajectory in O_Q enters E_Q and ψ_i^F from that point on Precisely stated, we have shown the following:

Proposition 4.6 Given a reachable task $i \in \mathbf{T}$ and an L_i^{*c}-restrictability compensator C_i so that i is consistent with C_i, if (1) the last event of each string in L_i is an

observable event, and (2) for all $\hat{z} \in E_Q$ and for all $(x, y) \in \hat{z}$, $x = x_f^{L_i}$ then task i is observable in A_{C_i}. ❑

The procedure explained above allows us to detect the first completion of task i. Detecting other completions of task i is straightforward: Suppose that O enters the state $\hat{y}$ when O_Q enters E_Q. Note that $\hat{y} \in E_o(L_i^{*c})$. At this point we detect the first occurrence of task i and in order to detect the next occurrence of task i, we immediately re-start O_Q at state $x_0^{L_i} \times \hat{y} \cap X_Q$. The procedure continues with each entrance into E_Q signaling task completion and a re-start of O_Q. Note that the observer O runs continuously throughout the evolution of the system. Let $D_i^* : \Gamma^* \rightarrow \{\epsilon, \psi_i^F\}$ denote the complete task detector system (which, for simplicity, assumes an initial observer state of $\{Y\}$). We can think of D_i^* as a combination of three automata: the observer O, the system O_Q which is re-started when a task is detected, and a single one-state automaton which has a self-transition loop, with event ψ_i^F, which occurs whenever a task is detected. This event is the only observable event for this system. Note that both the O_Q re-start and the ψ_i^F transition can be implemented as forced transitions.

Finally, in the same way in which we constructed C from the C_i, we can also define a task detector D from the set of individual task detectors D_i. Specifically, if C is set at C_i initially, D is set at D_i. Using the output Φ_T of C, D switches between D_i. For example, if D is set at D_i and τ_j^F is forced by C, then D switches to D_j. The output of D takes values in $\Gamma_T = \{\psi_1^F, \ldots, \psi_p^F\}$. A block diagram for D is illustrated in Figure 4.3.

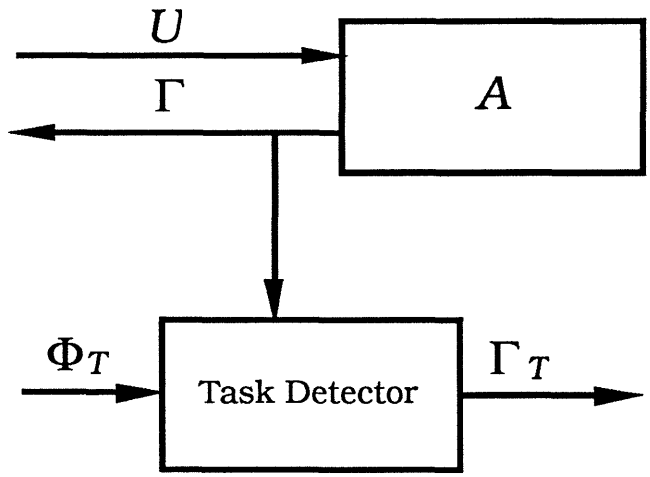

Figure 4.3: Task Detector Block Diagram

4.3 Task-Level Closed Loop Systems and Task Standard Form

Using the pieces developed in the preceding subsections we can now construct a task-level closed-loop system as pictued in Figure 4.4. The overall system is $A_{CD} = (G_{CD}, f_{CD}, d_{CD}, t_{CD}, h_{CD})$ where

$$G_{CD} = (X_{CD}, \Sigma \cup \Phi_T \cup \Gamma_T, \Phi \cup \Phi_T, \Gamma \cup \Phi_T \cup \Gamma_T, \Xi \cup \Phi_T) \tag{4.7}$$

Note that Φ_T and Γ_T are both observable and Φ_T is observable. Also, we include Φ_T in the tracking events to mark the fact that the system has switched compensators. This is important since following the switch, we will allow a finite length set-up. Also, since it does not make much sense in practice to force a switch to another compensator while the system is in the middle of completing a task, we impose the restriction that events in Φ_T can only be forced right after a task is completed. Since we require that all the tasks are observable (see Proposition 4.7), we can easily implement this

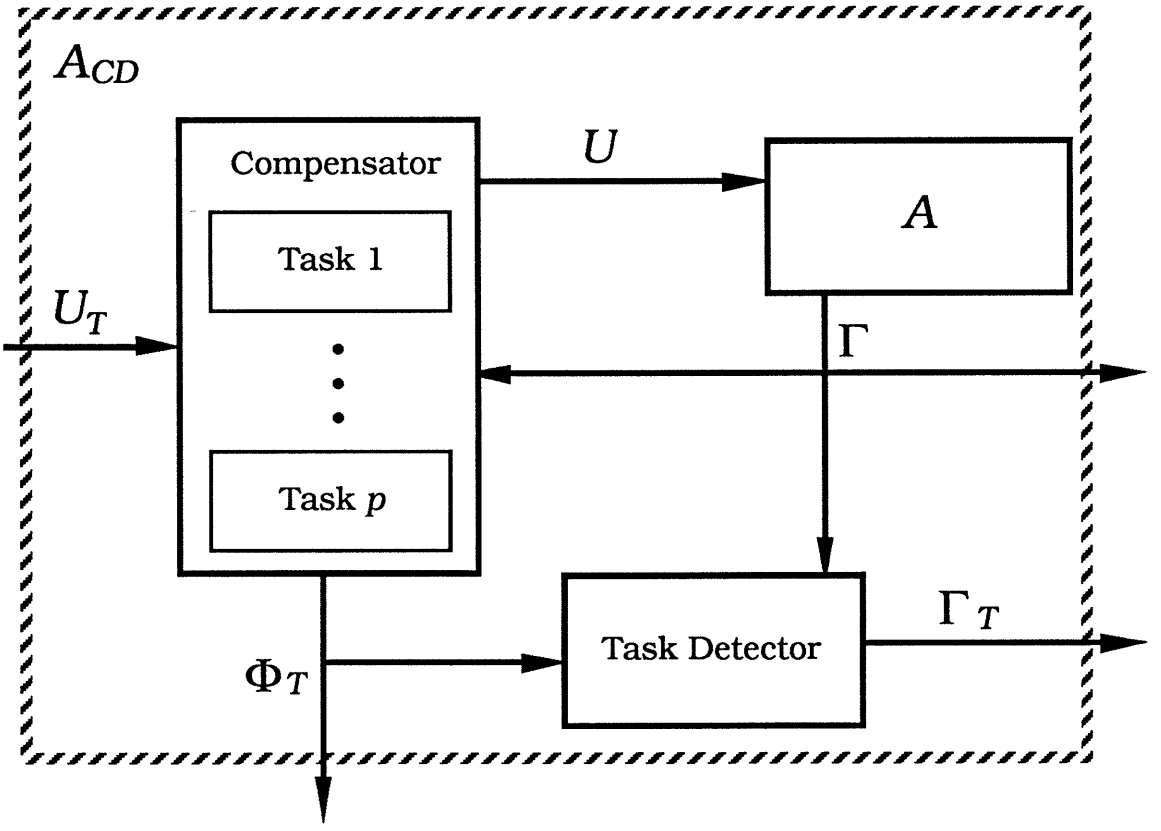

Figure 4.4: The Task-Level Closed-Loop System

restriction. Then, A_{CD} can only generate strings s such that

$$t(s) \in (\Xi \cup \{\epsilon\})^{n_t}(L_1^* \cup \cdots \cup L_p^*)(H_e(\tau_1)L_1^* \cup \cdots \cup H_e(\tau_p)L_p^*)^*$$

where n_t is the maximum number of tracking transitions needed until O enters the set of recurrent states in $E_o(L_i^{*c})$ for each $i \in \mathsf{T}$.

The higher-level operation of this system consists of the task initiation commands, Φ_T and the task completion acknowledgements, Γ_T. The input U_T indicating what subset of tasks can be enabled can be thought of as an external command containing the choices of subsets of Φ_T to be enabled. The use and control of this command involves higher-level modelling or scheduling issues beyond the purely task-level concept. What we show in this section is that the task-level behavior of A_{CD} can in fact be modelled, in the precise sense introduced in Section 3, by a much simpler automaton $A_{TSF} = (G_{TSF}, f_{TSF}, d_{TSF})$ illustrated in Figure 4.5 where all the events are controllable and observable, i.e.,

$$G_{TSF} = (X_{TSF}, \Sigma_{TSF}, \Phi_{TSF} = \Sigma_{TSF}, \Gamma_{TSF} = \Sigma_{TSF}) \tag{4.8}$$

We are not concerned with defining the tracking events of A_{TSF} since this alphabet is are not of concern in our main result below. We term A_{TSF} the task standard form.

Let us first define H_e. We first define $H_e(\epsilon) = \epsilon$ and $H_e(\psi_i) = L_i$. Note that, thanks to the independence of T, for any pair of not necessarily distinct tasks i and j, no suffix of string in $H_e(\psi_i)$ can be in $H_e(\psi_j)$. Defining $H_e(\tau_i)$ is more tricky. There are two issues:

1. We need to take into account the fact that the closed loop system does not generate strings in L_i immediately after C switches to C_i. In particular, if we assume that O is in a recurrent state when C switches to C_i and if we let n_e

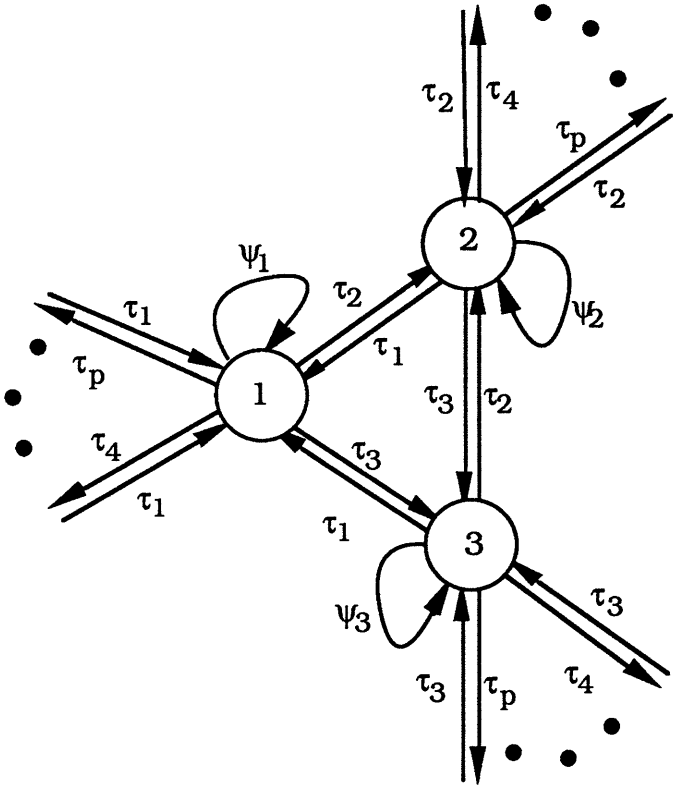

Figure 4.5: Task Standard Form: All events are controllable and observable.

denote the maximum number of tracking transitions that can occur in A for any trajectory in O that starts from a recurrent state of O up to and including the transition that takes the trajectory to a state in $E_o(L_i^{*c})$, then at most n_e tracking transitions can occur after C switches to C_i and before the behavior of the closed loop system is restricted to L_i^{*c}. Thus, we must choose H_e so that $H_e(\tau_i) \subseteq \tau_i^F(\Xi \cup \{\epsilon\})^{n_e}$.

2. We also need to ensure the minimality of H_e. Specifically, we now know that $H_e(\tau_i) \subseteq \tau_i^F(\Xi \cup \{\epsilon\})^{n_e}$. Suppose that we let $H_e(\tau_i) = \tau_i^F(\Xi \cup \{\epsilon\})^{n_e}$. Then, no suffix of a string in $H_e(\psi_i)$ can be in $H_e(\tau_i)$ since all strings in $H_e(\tau_i)$ start with τ_i^F. Also, no suffix of a string in $H_e(\tau_i)$ can be in $H_e(\tau_j)$ even if $i = j$. However, a suffix of a string in $\tau_i^F(\Xi \cup \{\epsilon\})^{n_e}$ may be in $H_e(\psi_j)$ for some j. Thus, we let $H_e(\tau_i) = (\Xi \cup \{\epsilon\})^{n_e} \cap \overline{(\Xi \cup \{\epsilon\})^{n_e} L_T}$. Note that thanks to consistency, the strings in L_T cannot occur in a set-up of a task. Therefore, eliminating strings that end with a string in L_T will not cause any problems in restrictability.

Proposition 4.7 Given a set of tasks **T** that is reachable by output feedback and observable, A_{TSF} is an H_e-model of A_{CD}.

Proof: We first verify the detectability condition. Before defining H_o, let us define $t' : \Phi_T \cup \Gamma_T \rightarrow \Sigma_{TSF}$ as $t'(\tau_i^F) = \tau_i$ for all i and $t'(\psi_i^F) = \psi_i$ for all i. We then pick H_o as t' of the projection of the observation sequence over $\Gamma \cup \Phi_T \cup \Gamma_T$ to $\Phi_T \cup \Gamma_T$, i.e., $H_o(s) = t'(s \downarrow (\Phi_T \cup \Gamma_T))$, where, $s \downarrow \Pi$, in general, denotes that part of the string s over the alphabet $\Pi \subset \Sigma$. Finally, let n_d be n_t divided by the length of the shortest string in L_T. Then, thanks to observability, the first detectability condition is satisfied. Also, using minimality it is straightforward to verify the second detectability condition.

To verify the restrictability condition, we proceed as follows: Note that A_{TSF} is

eventually restrictable to any infinite length string s in $L(A_{TSF})$. Thus, if we show that A_{CD} is eventually $H_e(s)^c$-restrictable by output feedback, then the restrictability condition is verified. Let us now proceed with showing this. If the first event of s is some τ_i, then we simply force τ_i^F and look at the second event. If the first event of s is some ψ_i, then we force τ_i^F and wait until ψ_i. When ψ_i occurs, we look at the second event. In both cases, when we look at the second event, we repeat the same process. It then follows that A_{CD} is restricted as desired. Therefore, A_{TSF} is an H_e-model of A_{CD}. ❑

5 High-Level Models of Composite Systems

The formalism described in the previous section can obviously be applied in an iterative fashion to obtain a hierarchy of aggregate models in which words (i.e., tasks) at one level are translated into letters (higher level events) at the next level. In addition to such a vertical configuration of such models, it is also of considerable interest to investigate horizontal configurations. Specifically, in many applications the overall DEDS is actually an interconnection of a number of simpler DEDS. For example, a flexible manufacturing system (FMS) can be viewed as an interconnection of workstations and buffers. In such a system, typically the system-wide tasks to be performed can be broken down into individual tasks performed by subsystems. In this case, it makes sense to develop individual task controllers for each subsystem and then consider their interconnections, and obviously the computational savings from such an approach may be considerable. In this section we develop conditions under which we can construct such a system-wide task-level model from local task models. Once we have such an aggregate system-wide model one can then consider higher-level coordinated control of the entire system, and this we illustrate through a simple model of an FMS.

5.1 Composition

In this section we focus on a simple class of models. Specifically, we wish to examine a DEDS that consists of a set of uncoupled subsystems, i.e., as described by automata with no shared events. For example, a collection of workstations can be thought of in this manner. Obviously, for an FMS to produce anything useful, it will need to coordinate the activity of these workstations, e.g., by connecting them with conveyor

belts, including buffers, etc. However, such coordination affects behavior at the task level—i.e., it does not influence how a workstation performs a specific task but rather what sequence of tasks is performed by what sequence of workstations . Thus, such coordination is in essence a higher-level control which can be viewed as a restriction on the behavior of the composite system. In the next subsection we will illustrate how such coordination can be incorporated, and this allows us to focus here on the uncoupled behavior of the subsystems.

Suppose that our system has m subsystems and that there is a set of reachable and observable tasks, $\mathbf{T}^j$ defined for each subsystem j. Let $A^j_{TSF} = (G^j_{TSF}, f^j_{TSF}, d^j_{TSF})$ be the task standard form for subsystem j, where Σ^j_{TSF} consists of τ^j_i and ψ^j_i for all tasks $i \in \mathbf{T}^j$. Then, the (uncoupled) interconnections of these subsystems can be simply represented by

$$A^C = (G^C, f^C, d^C, h^C, t^C) = \|^m_{j=1} A^j_{CD} = A^1_{CD} \parallel \cdots \parallel A^m_{CD} \tag{5.1}$$

If we also let

$$A^{SF} = (G^{SF}, f^{SF}, d^{SF}, h^{SF}, t^{SF}) = \|^m_{j=1} A^j_{TSF} \tag{5.2}$$

and we let

$$H_e(\sigma) = H^j_e(\sigma), \text{ for } \sigma \in \Sigma^j_{TSF} \tag{5.3}$$

then since A^j_{TSF} share no events, H_e is minimal.

In order to state the main result of this section we need the following, where $A_\emptyset$ denotes A with all controllable events disabled:

Definition 5.1 Task $i \in \mathbf{T}$ is preventable if for all $s \in L(A_\emptyset)$ $t(s) \notin L_i$. $\mathbf{T}$ is preventable if all its tasks are preventable. ❑

In the rest of this section, we will assume that $\mathbf{T}^j$ is preventable for all j:

Proposition 5.2 A^{SF} is an H_e-model of A^C.

Proof: Let us first verify the detectability condition. We let Φ_T^j and Γ_T^j denote the sets Φ_T and Γ_T of subsytem j. As in the proof of Proposition 4.7, we define a function t' so that $t'(\tau_i^{Fj}) = \tau_i^j$ and $t'(\psi_i^{Fj}) = \psi_i^j$ for all i and j. Then, we let

$$H_o(s) = t'(s \downarrow (\bigcup_{j=1}^{m} \Phi_T^j \cup \bigcup_{j=1}^{m} \Gamma_T^i))$$

and the rest of this proof follows as in the proof of Proposition 4.7.

To verify the restrictability condition, we proceed as follows: Note that A^{SF} is eventually restrictable to any infinite length string s in $L(A^{SF})$. Thus, if we show that A^C is eventually $H_e(s)^c$-restrictable by output feedback, then the restrictability condition is verified. Let us now proceed with showing this. If the first event of s is some τ_i^j, then we force τ_i^{Fj}, disable all events of all the other subsystems and look at the second event. If the first event of s is some ψ_i^j, then we force τ_i^{Fj}, disable all events of all the other subsystems and wait until ψ_i^j. When ψ_i^j occurs, we look at the second event. In both cases, when we look at the second event, we repeat the same process. Also, thanks to preventability, it then follows that A^C is restricted as desired. Therefore, A^{SF} is an H_e model of A^C. ❑

The concept of preventability and the nature of our higher level model deserve some comment. The framework we have described here is essentially one of serial operation—i.e., our system-wide task level model describes the sequence in which tasks are completed by all subsystems (i.e., task 1 by workstation 1, then task 3 by workstation 2, ...). While preventability is an essential concept in any system, in practice we only want to prevent a subsystem from operating if its next action is truly serially dependent on the completion of another task by another subsystem. In other cases we may want to allow several subsystems to be operating in parallel. While it

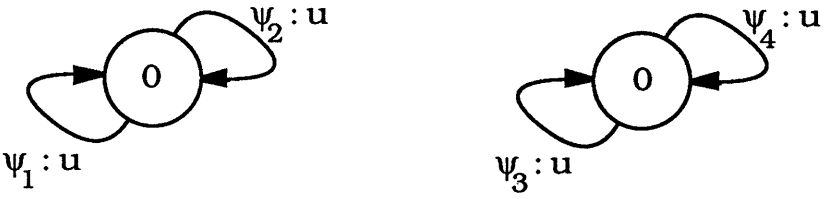

Figure 5.1: Two Systems

is possible to infer parallelism by a detailed examination of event and task sequences (much as one can when examining the event trajectory for the shuffle product [10] of susbsystems), this is not a totally acceptable solution. To capture parallelism in a more direct way. we need to add complexity to our task standard form by keeping track not only of task completion but also task initiation, so that in the task-level composite our state at any time will indicate what set of tasks are ongoing. The detailed development of these ideas is left to a future paper.

5.2 Subsystem Interactions and Higher Level Control

As we indicated previously, interactions between subsystems can often be modelled via restrictions on the system-wide task-level model. In the following example, we illustrate how the presence of buffers between workstations can be represented as such restrictions. Furthermore, we will see that the restrictions imposed by each buffer can be dealt with independently, and each restriction can be viewed as the action of a compensator.

Example 5.3 Suppose that we have two systems, each capable of performing two reachable and observable tasks. We let task 1 and 2 be the tasks associated with the first system, and tasks 3 and 4 be the tasks associated with the second

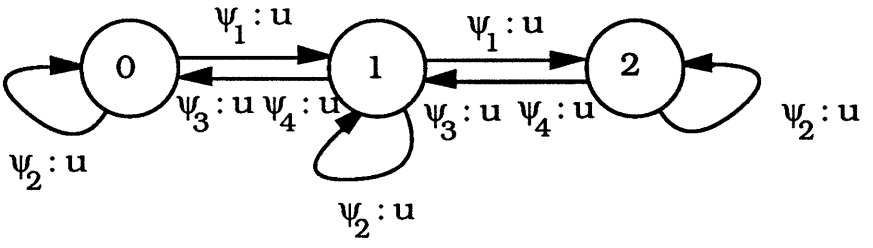

Figure 5.2: First Buffer Implementation

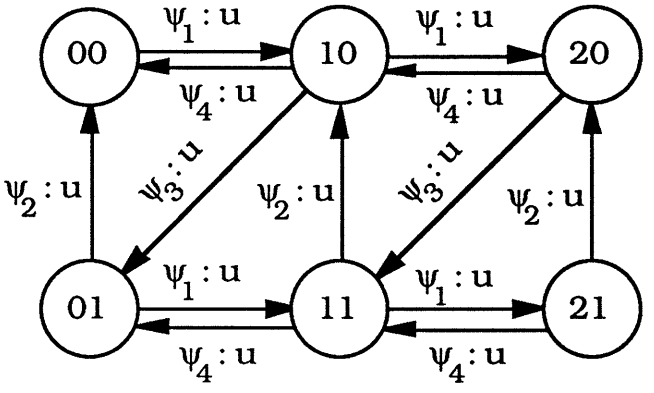

Figure 5.3: Second Buffer Implementation

system. We model these systems by the task-level automata illustrated in Figure 5.1 where all tasks are also assumed to be observable and for simplicity of exposition, we have assumed that there is no set-up involved in switching between tasks (i.e., the workstations produce no extraneous events between the event sequences corresponding to the various tasks). In this case there is no need to include both ψ_i and τ_i in the task-level model. Suppose that each system is a model of a workstation in an FMS, which manufactures two different parts, so that the first part can be manufactured by performing tasks 1, 3, and 2 successively, and the second part can be manufactured by performing task 1 followed by task 4. Let us assume that there is a buffer of size two between the two workstations so that every occurence of ψ_1 increases this buffer and every occurence of ψ_3 or ψ_4 decreases it. This buffer imposes two restrictions on the system: First, ψ_1 should not occur when the buffer is full, and second, ψ_3 and ψ_4 should not occur when the buffer is empty. These restrictions can be implemented in a straightforward way, and the closed loop system is illustrated in Figure 5.2. Let us also assume that there is a buffer of size one between the two workstations so that every occurence of ψ_3 increases this buffer and every occurence of ψ_2 decreases it. Restricting the system further by the conditions imposed by this buffer yields the closed loop system in Figure 5.3. ❑

In general, given A^C, A^{SF}, and H_e so that A^{SF} is an H_e-model of A^C, suppose that a buffer restriction can be implemented by a compensator C on A^{SF}. Then, thanks to Proposition 3.5, we can find a compensator C' so that the closed loop system A^{SF}_C is an H_e-model of the closed loop system $A^C_{C'}$. Furthermore, we can consider further restrictions on the behavior of A^{SF}_C corresponding to higher-level primitives, called

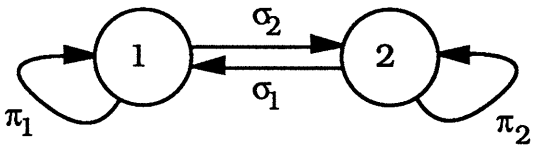

Figure 5.4: Procedure Standard Form for Example 5.3

procedures, which consist of sequences of tasks. Let us illustrate this for Example 5.3. In this example, we let A^C_{SF} represent the system in Figure 5.3. Based on the two parts to be manufactured defined in Example 5.3, we define two procedures for this system, namely $L_1 = \psi_1\psi_3\psi_2$, and $L_2 = \psi_1\psi_4$. It is straightforward to show that the states 00 and 10 are L_1^{*c}-restrictable, and states 00,10,01, and 11 are L_2^{*c}-restrictable, and furthermore, A^C_{SF} is both eventually L_1^{*c}-restrictable and eventually L_2^{*c}-restrictable. Note that since all tasks are observable, we do not need to worry about procedure detection in this case. Thus, we can construct compensators for each procedure as we did for tasks in the previous section. Let K denote the combined compensator, let A^{SF}_{CK} denote the closed loop system, and let A_{PSF} denote the automaton in Figure 5.4, where σ_i represents switching to procedure i and π_i represents completing procedure i. Then we can find a map H_e so that A_{PSF} is an H_e-model of A^{SF}_{CK}. We term A_{PSF} the procedure standard form. Finally, thanks to Proposition 3.6, A_{PSF} also models $A^C_{C'K}$, our original system combined with task compenators for each system, buffer restrictions implemented by C', and the procedure compensator K.

6 Conclusions

In this paper, we have introduced concepts of higher-level modelling for DEDS based on a given set of primitive event sequences corresponding to tasks which the system may perform. Through our investigation of task reachability we constructed task compensators and this, together with task observability allowed us to construct simple high-level models of automata so that events in the high-level model correspond to set-up and completion of tasks. We have also considered the higher level modelling of systems that are composed of a set of subsystems and we have shown how the coordinated control of such a system can be effected by higher-level control which we have referred to as procedures.

The aggregation scheme presented in this paper addresses an important issue of computational complexity in DEDS problems of interest. In particular, for computations involving systems that are composed of m subsystems so that the description of each subsystem has n states, the complexity is a function of n^m. Our aggregation procedure is important in reducing n considerably since the cardinality of the state space of the high-level model equals the number of tasks.

Finally, there are several important directions for further research. As we have indicated, one of these involves the extensions of our approach in order to capture parallelism in interconnected systems. A second deals with a more careful examination of the information and control required at various levels of a hierarchical control system of the type described here. In particular, while complete control and observation of each task is essential at the individual subsystem level, such detailed information and control action may not be needed at the procedure level. For example, if a procedure consists of performing task 1 on machine 1 and then either task 7 on

machine 2 or task 5 on machine 3, the procedure-level system does not need to know which alternative has been followed but only needs an acknowlegement that one of them has been completed. By developing a rational methodology for minimizing the diversity of control actions and information required at any level we may be able to reduce considerably the information being transmitted and stored in the overall system. For example, this is of obvious importance in communication systems, since the commands to perform tasks such as transmitting packets and information on the completion of such tasks need to be transmitted through the network, and as these control transmissions increase, the network performance deteriorates.

References

[1] J. S. Ostroff and W. M. Wonham. A temporal logic approach to real time control. In *Proceedings of CDC*, December 1985.

[2] C. M. Özveren and A. S. Willsky. Invertibility of discrete event dynamic systems. in preparation.

[3] C. M. Özveren and A. S. Willsky. Observability of discrete event dynamic systems. Submitted to the *IEEE Transactions on Automatic Control.*

[4] C. M. Özveren and A. S. Willsky. Output stabilizability of discrete event dynamic systems. in preparation.

[5] C. M. Özveren and A. S. Willsky. Tracking and restrictability in discrete event dynamic systems. in preparation.

[6] C. M. Özveren, A. S. Willsky, and P. J. Antsaklis. Stability and stabilizability of discrete event dynamic systems. Submitted to the *Journal of the ACM.*

[7] P. J. Ramadge and W. M. Wonham. Modular feedback logic for discrete event systems. *SIAM J. of Cont. and Opt.*, September 1987.

[8] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. of Cont. and Opt.*, January 1987.

[9] A. F. Vaz and W. M. Wonham. On supervisor reduction in discrete event systems. *International Journal of Control*, 1986.

[10] W. M. Wonham. A control theory for discrete-event systems. In *NATO ASI Series, Advanced Computing Concepts and Techniques in Control Engineering.* Springer-Verlag, 1988.