4

Design of Structure-Based Hierarchies for Distributed Intelligent Control

Levent Acar
Dept. of Electrical Engineering
University of Missouri-Rolla
Rolla, MO 65401

Ümit Özgüner
Dept. of Electrical Engineering
The Ohio State University
Columbus, OH 43210

Abstract

In this chapter, intelligence is embedded in control via a special hierarchical organization based on the physical structure of the system. The organization is inherently object-oriented, and the control and the knowledge are distributed throughout the hierarchy. The hierarchy provides a multi-resolutional, distributed and overlapping representation of the system, where each node of the hierarchy contains some level of mathematical description, intelligence and local (sub)optimal control. Theoretical foundations of these intelligent controllers are presented in an axiomatic approach which mathematically describes the hierarchy, its functionality and the control process. This approach also provides formal definitions for concepts such as abstraction, summarizing and decomposition.

1. INTRODUCTION

In recent years, the incorporation of intelligence into control systems has been focused on two major approaches. One of these approaches organizes the intelligence in planning, reasoning and control hierarchically. The other approach creates intelligent response through a union of stimulus-response controllers. Although each approach has certain advantages, a deeper understanding of

planning and reasoning can be obtained from the hierarchical approach with greater ease, [1].

Hierarchical organizations can also be classified by the method by which they have been generated. One very popular method is to explore the functionality of the system, and to form the hierarchy based on the functional decomposition of the goals to be accomplished. In this chapter, this type of hierarchy will be referred as the function-based hierarchy. The other method is based on the objects associated with the physical structure of the system. These objects may be components of the system, or they may be combinations of parts of the system. To be general, throughout the chapter, this type of object-oriented hierarchy will be referred as the structure-based hierarchy.

In control, function-based hierarchies have been utilized for more than a decade with great success, [2–12]. There are numerous examples where controls show intelligent decision making capabilities to accomplish desired goals. Implementational details of these hierarchies are provided in various chapters of this book. However, most of these hierarchies lack strong mathematical bases, and analytical results about their properties and behaviors may be only partially obtainable, [13].

In this chapter, a structure-based hierarchy will be described, its mathematical foundation and theory will be presented, mathematical bases of loosely defined concepts, such as abstraction, decomposition, etc., will be given, and intelligent distributed controllers will be introduced. Feedback methods for real-time execution will also be discussed. Examples will be provided to demonstrate the development of the structure-based hierarchical controllers.

2. STRUCTURE-BASED HIERARCHY

To obtain a structure-based hierarchy, we will concentrate on physical systems. At this point, the desired behavior of the system and its functionality are not considered. This starting point is the major difference between function-based and structure-based hierarchies. By concentrating on the physical system initially, a very large set of its functionality could be explored later. In contrast, function-based hierarchies concentrate initially on the goal to be accomplished by a system, which enables exploration of a large set of systems for a limited number of goals.

2.1. Obtaining the Nodes of the Hierarchy

The first step in a structure-based hierarchy is to identify the nodes. Since this hierarchy is based on the physical system, the nodes are composed of portions of the physical system. An initial set of system components or portions is assumed to be available. This set could consist of the control designer's choice of actuators, sensors, or it could consist of arbitrary three dimensional slices of the system. Obviously, in the arbitrary-cut case, insight about the system components will be lost.

Mathematically, the given system is represented by Σ_0, and the initial sets by $\{\Sigma_i\}_{i\in\mathcal{Z}^+}$, where $\mathcal{Z}^+$ is the set of all positive integers. Usually, the set $\{\Sigma_i\}_{i\in\mathcal{Z}^+}$ is finite. The cases in which the set is infinite, the hierarchy may have infinitely many nodes.

Usually initial choices have less than necessary nodes to design the hierarchy. However, as you will observe later, the initial choice determines the granularity of the hierarchy and is important.

To obtain more nodes for the hierarchy, at least the unions, intersections and complements of the sets $\{\Sigma_i\}_{i\in\mathcal{Z}^+}$ need to be included. Since, topological properties and measurability of the spaces obtained by these sets will be also needed, more general sets need to be defined. Let $\mathcal{T}_{\boldsymbol{\Sigma}}$ be a topology in Σ_0 such that $\{\Sigma_i\}_{i\in\mathcal{Z}^+} \in \mathcal{T}_{\boldsymbol{\Sigma}}$. Therefore, the initial choice, all finite intersections, and unions of the initial choice are open sets in Σ_0. Also, we form the Borel set $\mathcal{B}_\Sigma$ in Σ_0, and denote the sets in $\mathcal{B}_\Sigma$ by Σ_s. The inclusion of measurability enables us to be able to define mappings which would map open sets to open sets. The open sets in $\mathcal{B}_\Sigma$ will form the nodes of the hierarchy.

2.2. Organizing the Hierarchy

After the possible nodes of the hierarchy is determined, the second step is to connect these nodes in a coordinated way. The connection of these nodes will be done in such a way that the system will be described repeatedly in different detail in the hierarchy. The advantages of this type of an organization are many. First, it enables different system and environment representations to exist simultaneously. Second, it permits various levels of time granularity and a planning horizon. Third, it provides deep reasoning[1] and planning. And finally, it improves failure detection and handling techniques. Some of these advantages will be clear as the development of the hierarchical controllers progresses in the next few sections. Moreover, this concept is further explored in [14].

To be able to formally describe this type of hierarchy, first define an index set to describe the relations among the nodes.

Definition 2.1: $\mathcal{I}_s^{[n]}$ *is a set of node indices, such that*

$$\mathcal{I}_s^{[n]} = \begin{cases} \{\, i \mid \Sigma_i \text{ is a } -nth \text{ supernode of } \Sigma_s \,\}, & n \in \mathcal{Z}^- \\ \{\, s \,\}, & n = 0 \\ \{\, i \mid \Sigma_i \text{ is a } nth \text{ subnode of } \Sigma_s \,\}, & n \in \mathcal{Z}^+ \end{cases},$$

where a nth supernode of Σ_s is a node which reaches Σ_s via n links of the directed graph, and similarly a nth subnode of Σ_s is a node which is reached from Σ_s via n links.

Here, $\mathcal{Z}^-$ represents the negative integers. Finally, the type of hierarchical connections is described through the following definition using an axiomatic approach.

[1] As in Artificial Intelligence context.

Definition 2.2: *A directed graph with no structural loops, whose nodes are the open sets in $\mathcal{B}_\Sigma$ is said to be* structurally coordinated, *if it satisfies the following* Structural Coordinability Axioms.

Structural Coordinability Axioms

1. $\Sigma_0 \neq \emptyset$, *and* $\mathcal{I}_0^{[-1]} = \emptyset$.
2. $\Sigma_s \cap \Sigma_j \neq \emptyset, \quad \forall j \in \mathcal{I}_s^{[1]}$.
3. $\Sigma_s \subseteq \bigcup_{j \in \mathcal{I}_s^{[1]}} \Sigma_j$.
4. $\Sigma_j \in S_{\max}, \quad \forall j : \mathcal{I}_j^{[1]} = \emptyset,$
   where $S_{\max} \in \mathcal{S} = \{ S \mid S = \{ \Sigma_i \mid i \in \{0, 1, \ldots\},$
   Σ_i*'s are distinct* $\} \}$,
   and $\overline{\overline{S}}_{\max} \geq \overline{\overline{S}}, \quad \forall S \in \mathcal{S}$.

Here, $\overline{\overline{S}}$ *represents the cardinal number of the set* S.

There may be numerous structurally coordinated hierarchies derived from the same initial choice of sets, and one may be faced with a choice. This choice will be considered in greater detail in the next section, where functionalities are assigned to the nodes.

Although, the *Structural Coordinability Axioms* do not provide a unique organization for a given initial choice of sets, they structure the hierarchy in a very interesting way. First of all, as a result of the first axiom, the single-node representation of the whole system is placed at the top. This top node indeed gives the least detailed representation of the system with the largest horizon and the most coarse time granularity. The second axiom guarantees a common portion between a node and any of its subnodes. Moreover, the third axiom ensures that the collection of the subnodes of a node should be at least as large as the node itself. These two axioms shape the hierarchy to represent the whole system over and over again with greater detail. As a result, the time granularity gets finer as the horizon or scope gets shorter. Finally, the last axiom forces the tip level to consist of the maximum number of disjoint nodes. Therefore, the tip level has the most detailed representation possible given the initial choice of sets.

The following theorems further explore the consequences of the *Structural Coordinability Axioms*. The proofs of these theorems are included in the appendix.

Theorem 2.1: *The whole system is described completely at the tip level, i.e.,*

$$\Sigma_0 = \bigcup_{j : \mathcal{I}_j^{[1]} = \emptyset} \Sigma_j .$$

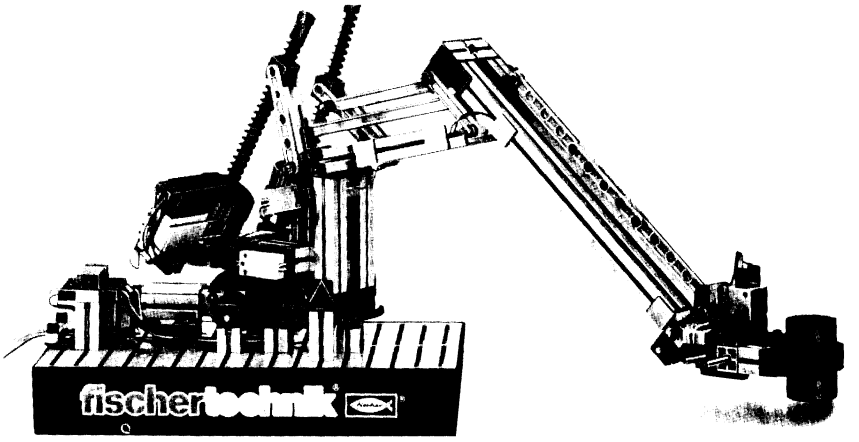


Figure 2.1: Fischertechnik® robot arm.

Theorem 2.2: *The tip level has the finest decomposition, i.e.,*

$$\Sigma_i \cap \Sigma_j = \emptyset \quad or \quad \Sigma_i \cap \Sigma_j = \Sigma_j, \quad \forall \Sigma_i \in \mathcal{B}_\Sigma, \quad and \quad \forall j : \mathcal{I}_j^{[1]} = \emptyset.$$

Theorem 2.3: *Any two nodes which are not subnodes of each other and have some common portions have to have common subnodes in which the common portion is included, i.e.,*

$$\Sigma_s \cap \Sigma_t \neq \emptyset, \qquad t \notin \mathcal{I}_s^{[n]}, \quad \forall n \in \mathcal{Z}$$

$$\Longrightarrow \Sigma_s \cap \Sigma_t \subseteq \bigcup_{i=1}^{k} \Sigma_{m_i}, \qquad \exists k, m_1, p_1, q_1, \ldots, m_k, p_k, q_k \in \mathcal{Z}^+$$

$$: m_i \in \mathcal{I}_s^{[p_i]} \cap \mathcal{I}_t^{[q_i]}, \quad \forall i = 1, \ldots, k.$$

2.3. Applying the Hierarchy

A robotic manipulator is a good example for the application of a structure-based hierarchy. Physically, it is complicated enough; functionally, it may exhibit intelligence; and it is also a challenge to control. A robotic manipulator could be simple in appearance like the the fischertechnik® robot arm, shown in Figure 2.1, or it could be more complicated like the Intelledex manipulator, shown in Figure 2.2.

Depending on the initial choice, many structurally coordinated hierarchies can be obtained from these systems. Examples for each of the robotic manipulators are given in Figures 2.3 and 3.4. In these figures, the dashed boxes

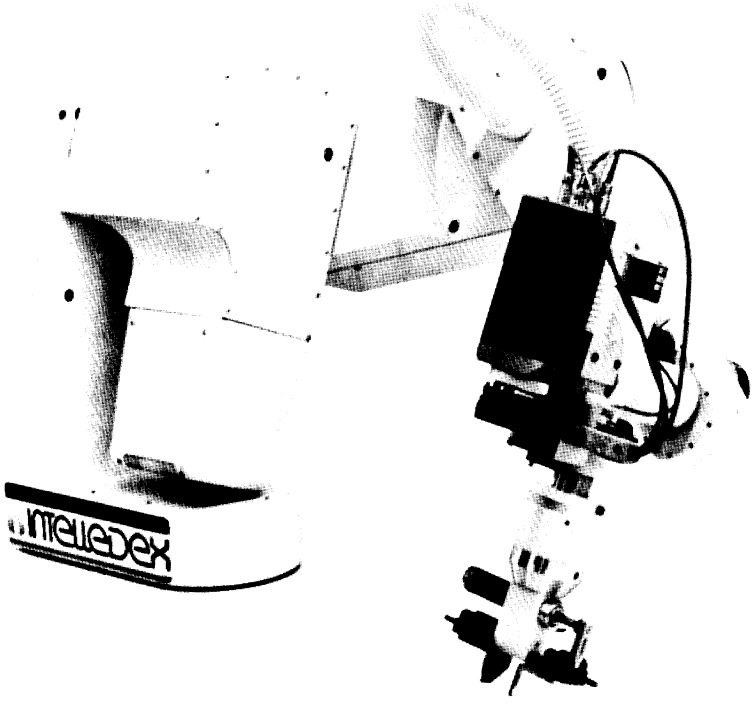


Figure 2.2: Intelledex Model 605T robotic manipulator.

represent the initial choices. However, it should also be noted that even with the same initial choice set, a different hierarchy may be obtained depending on the choice of the other nodes. Indeed, the structural coordinability axioms do not even force us to include the sets of the initial choice in the hierarchy. This issue of obtaining multiple structurally coordinated hierarchies will be discussed further in the next section, where functionalities are assigned to the nodes.

3. FUNCTIONALITY ASSIGNMENT

A structurally coordinated hierarchy requires functionality and a control process to accomplish a desired goal. In this section, a method to embed functionality which will be the basis for the control process of the next section will be developed.

Initially, one might think that the assignment of functionalities to each of the nodes of the hierarchy is fairly easy. This would have been easy, if one didn't

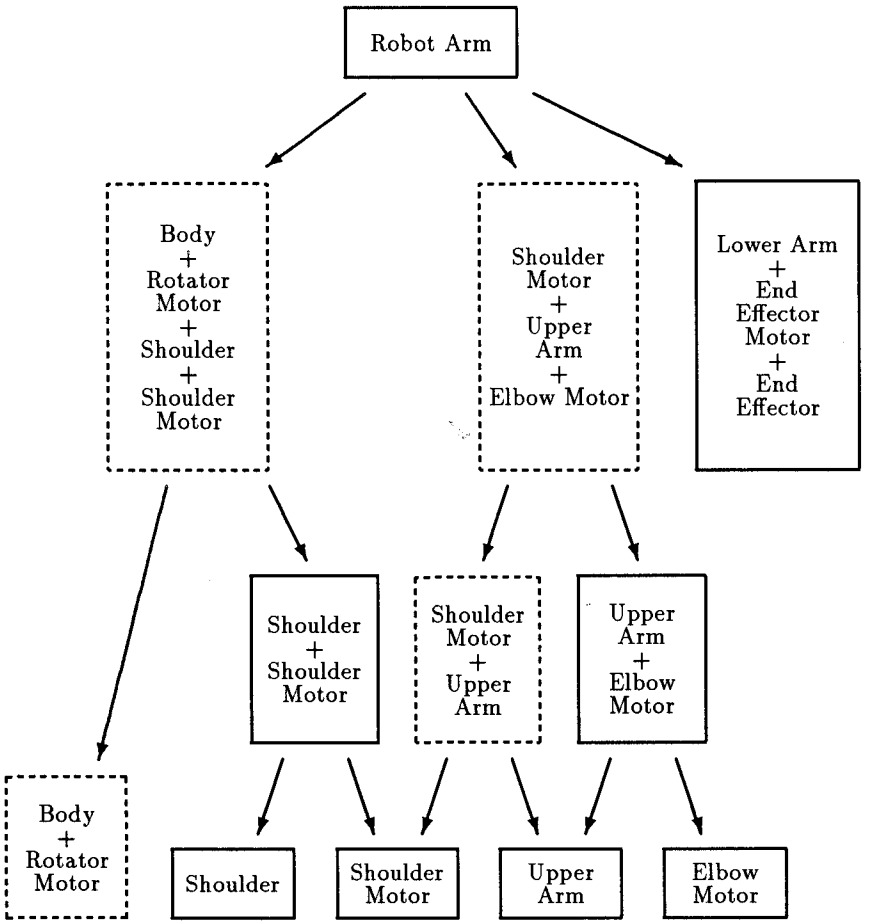


Figure 2.3: One possible structurally coordinated hierarchy for the Fischertechnik® robot arm.

have to communicate or delegate work throughout the hierarchy. Realizing that the job is two-fold: the first being the local functionality assignment, the second the communication among the nodes; one approach is to separate the two and to solve one problem at a time.

3.1. Incorporating Intelligence to the Nodes

To represent the knowledge at a node, i.e., to assign its functionality, describe its accomplishable *tasks*, and the methods or *procedures* to accomplish them. This approach introduces two important concepts: tasks and procedures. In accomplishing these tasks, procedures frequently use outputs of sensors or *measurements*, and they are restricted by some *constraints* and perhaps limited

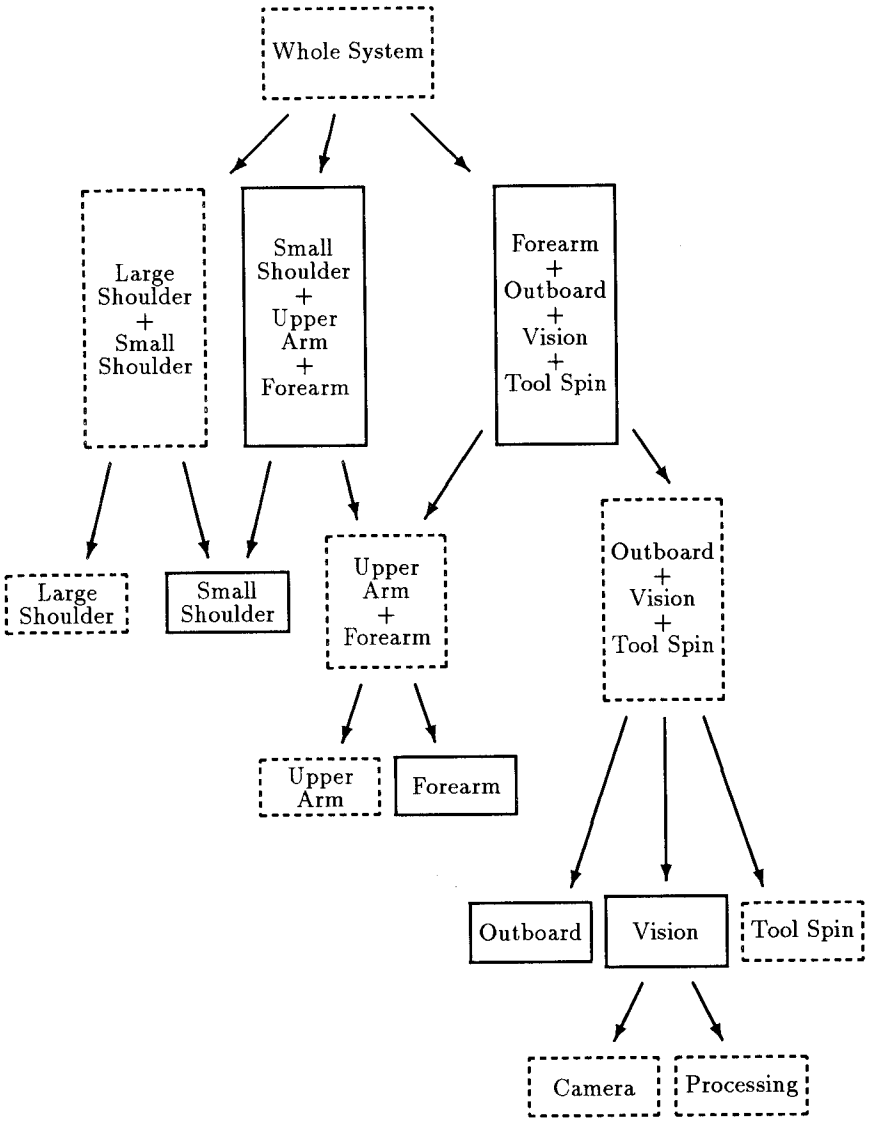


Figure 3.4: One possible structurally coordinated hierarchy for the Intelledex Model 605T robotic manipulator.

by some sort of *resources*. Therefore, the use of procedures introduces three more concepts: measurements, constraints and resources. These concepts will be the primitives on which the intelligence within the nodes of the hierarchy will be built. These primitives also incorporate the four of the seven basic elements, *actuators*, *sensors*, *sensory processing* and *world modeling*, of the model mentioned in [15]. Moreover, they are totally consistent with the *nested multiresolutional* approach discussed in [14], since the nodes of the hierarchy are organized in a multi-resolutional fashion.

To communicate or delegate work throughout the hierarchy, another concept will be used; the concept of setting goals for the subnodes. Goals and a few other primitives will be part of the control process in the next section.

Before all the six primitives are related with each other, brief descriptions with respect to the node s are given below.

Goals (G_s) are assertions which represent the end result to be obtained at node s.

Tasks (T_s) are elementary job descriptions at node s.

Procedures (Π_s) are methods of accomplishing tasks. The procedures are separated into two groups depending on their applicability.

1. *Local Procedures*[2], (Π_{s0}) are those procedures which are locally applicable at the node s.
2. *Subprocedures*, (Π_{s+}) are those procedures which are applicable by subnodes.

Measurements (M_s) are the available information from sensors. The measurements are also separated into two types.

1. *Local Measurements*, (M_{ss}) are the measurements available locally at node s.
2. *Other Measurements*, (M_{sj}) are the measurements of other nodes. Notationally, the jth node's measurements, which are available to the sth node, are denoted by M_{sj}.

Constraints (Γ_s) are task restrictions or exemptions. They are separated into two types as well.

1. *System Dependent Constraints*, (Γ_{s0}) are the type of constraints which depend solely on the system. They do not change with the environment or the goals. An example is the maximum torque limitation at a joint of a manipulator due to its construction.

[2] In the earlier publications, including [16], all of the primitives, which are referred as *Local* here, were referred as *Current*.

2. *System Independent Constraints*, ($\Gamma_{s\_}$) are the type of constraints which does not depend on the system but depends on the environment or on the assigned goals. The angular limitation of a manipulator due to an obstacle is a system independent constraint.

Resources (Υ_s) are task restrictions or exemptions which are related to the use of procedures. Similar to measurements, resources are also separated into two groups:

1. *Local Resources*, (Υ_{ss}).
2. *Other Resources*, (Υ_{sj}).

As obvious from the above descriptions, some of the primitives are separated into two types. Let us consider them one by one, and explore the need to have different types of those primitives.

Procedures, as described above, are methods to accomplish tasks. As an example, if the task at a node is to apply a certain voltage to a port until a sensor reading reaches a given value, and if the node is capable of applying that voltage; then that procedure is called a local procedure. It merely means that the node can handle the task itself without asking help or delegating the job to another node. All the procedures at the tip nodes should be of this type, since they have no other node to delegate.

However, some of the tasks at the higher levels of the hierarchy may not be accomplished at that node. An example is the task of increasing the angle a certain amount at a joint of a manipulator. A higher node may have this task, and it may also have the knowledge that in order to increase the angle by that certain amount, five volts should be applied to the input port of the relevant motor, until eighty-five pulses are counted at its output port. The node realizing this solution, perhaps through one of its procedures, may not be able to apply the voltage itself directly, but it might also know that one of its subnodes is capable of doing just that. The procedure which encapsulates all this information is called a subprocedure.

The need to have subprocedures is essential in order to increase the processing speed of decision making in the hierarchy. Lack of this information would result in long search methods for even a simple task, and it would be against the intent of a hierarchy. This information may be incorporated as part of the subprocedures, or routines may be implemented for procedure inheritance which handles different representations among connecting nodes.

The real trade-off is in the completeness and reliability of this information. All the mathematical formulations presented in this section are based on the assumption that this information is complete and totally reliable. Corrections due to incomplete and incorrect information are incorporated during the replanning and failure handling stage.

Measurements are also separated into two types depending on their origin. The first type is the local measurements. It consists of all the measurements

locally available at each node. For the second type of measurements, the assumption is that measurements from other nodes may also be available at a node. This assumption is to remedy one of the disadvantages in controlling a non-centralized system. In a non-centralized system, all the information is usually not available everywhere, so control actions would have to be based on partial information. Enabling access to measurements everywhere eliminates this disadvantage and provides the same performance with the ease of a hierarchical design.

Although resources are similarly separated, the resources of other nodes will only be used for failure handling. Initially, the resources are assumed to be allocated at each node, but as the goals are communicated, they may also be upgraded.

The separation of constraints is a little bit different than the other separations. The system dependent constraints are static and fixed, and they are always applicable. On the other hand, the system independent constraints are dynamic constraints which change as the environment and goals change. The system independent constraints are usually associated with goals, tasks or procedures. They may define working spaces of goals and tasks. They may inhibit the activation of certain tasks providing an asynchronous behavior. They may also restrict application of certain procedures for certain tasks. The importance of constraints will be more clear when the control process is discussed.

At every node, there are some tasks, procedures, constraints, measurements and resources as part of its knowledge-base. Most of the primitives contain variables whose values will be supplied by the control process. In this chapter, the act of supplying specific values to variables will be referred as *instantiation*, and a primitive whose values are all instantiated will be referred as an *instance*. The contents of these primitives are given in more detail below.

Each task has a list of the following elements:

1. Variables to be instantiated.
2. Names of procedures which may accomplish it under certain constraints.
3. Routines which will decide the applicability of procedures under current constraints.
4. Routines which will instantiate applicable procedures.

Depending on the constraints, there might be more than one applicable procedure, and all of them should be instantiated for the same instance of a task. A selection process is needed to pick one, and this process is explained as part of the control process in the next section.

Procedures have the following elements:

1. Variables to be instantiated including the name of the task instantiating them.
2. Routines which apply control actions, or routines which decompose the instantiating task into smaller components for the subnodes.
3. Routines which recheck the constraints, and the use of resources.

As a result of the association among tasks and procedures, different instances of the same procedure might be activated for a task or for a bunch of tasks.

As mentioned earlier, the ordering of these procedures is achieved through the inclusion of constraints.

Constraints either exist independent of the tasks and procedures as most of the system dependent constraints do, or they are incorporated with them. The constraints, which are independent of tasks and procedures, are for the use of all the primitives and processes in the node. The other constraints, which are incorporated with the tasks and procedures, are for the specific primitives, and they are mostly derived due to the instantiation of the primitive. As an example for the latter case, think of the motion of the manipulator around an object. Assume that to accomplish the motion, a joint has to move thirty degrees in the positive direction first, and then after another joint reaches a precomputed value, it should move ten degrees in the negative direction. To accomplish these tasks, two instances of the same procedure might be chosen; one with $+30°$, and the other with $-10°$ for the angle variable. To accomplish the sequential behavior, a constraint in the second instance of the procedure is to be included, such that it doesn't initiate the motion, until the constraint is satisfied. The request for the status of the constraint variable may be performed by a routine in the procedure utilizing the other measurements primitive.

Measurements contain actual sensor values of the system. Measurements requested from other nodes, i.e., other measurements, are obtained via dynamically generated "smart" links. The links are established, when there is need, and they only relate the requested measurements. These links have only the local knowledge of the nodes they are linking. However, they are smart enough to translate one knowledge representation to another. In the example of the two instances of a procedure above, the second procedure with the constraint establishes a link to check the status of the other joint. These links also incorporate the seventh element, *global memory/communications* of the model mentioned in [15].

3.2. Formalizing Node Functionalities

Up to this point, only the pragmatic approach was discussed in describing functionality of nodes. Formal mathematical descriptions of these primitives are less intuitive, and they are based on the relations discussed above in the loosest sense. For example, as far as the mathematical representations are concerned, it is irrelevant where and how the measurements are obtained. The important mathematical property is the existence of an applicable procedure which will work under available constraints, measurements and resources. By including other measurements and "smart" links, an applicable procedure which utilizes the new measurements may be created, or the number of applicable procedures may be increased.

To be able to formally define the relations of primitives among the nodes of the hierarchy, domains of the primitives and some mappings are needed to be defined.

The domains for the primitives are defined first. Initial sets of all the primitives are chosen similar to the initial set of the structural coordination.

However, unlike the initial set of the structural coordination, the initial sets of primitives might not always lead to an acceptable functionality distribution in the hierarchy. Usually, this failure of acceptable distribution is due to the omission of some primitives, specifically some tasks and procedures. Therefore, all practically possible sets in the initial choice will be assumed included. As with all the axiomatic approaches, a method to generate initial sets of the primitives can not be proposed, but whether or not an acceptable functionality distribution is achieved for a given set can be checked.

For the initial sets of primitives, the sets of all the initial choices of *uninstantiated* goals, tasks, procedures, measurements, constraints and resources at node s are denoted by the sets $\{G_s^i\}_{i\in\mathcal{Z}+}$, $\{T_s^i\}_{i\in\mathcal{Z}+}$, $\{\Pi_s^i\}_{i\in\mathcal{Z}+}$, $\{M_s^i\}_{i\in\mathcal{Z}+}$, $\{\Gamma_s^i\}_{i\in\mathcal{Z}+}$ and $\{\Upsilon_s^i\}_{i\in\mathcal{Z}+}$, respectively. Then, all the primitives with all of their possible instances in the sets: $\mathbf{G_s}$, $\mathbf{T_s}$, $\mathbf{\Pi_s}$, $\mathbf{M_s}$, $\mathbf{\Gamma_s}$ and $\mathbf{\Upsilon_s}$[3] are included.

Furthermore, topologies in $\bigcup_\alpha\{G_s^\alpha \mid G_s^\alpha \in \mathbf{G_s}\}$, $\bigcup_\alpha\{T_s^\alpha \mid T_s^\alpha \in \mathbf{T_s}\}$, $\bigcup_\alpha\{\Pi_s^\alpha \mid \Pi_s^\alpha \in \mathbf{\Pi_s}\}$, $\bigcup_\alpha\{M_s^\alpha \mid M_s^\alpha \in \mathbf{M_s}\}$, $\bigcup_\alpha\{\Gamma_s^\alpha \mid \Gamma_s^\alpha \in \mathbf{\Gamma_s}\}$, and $\bigcup_\alpha\{\Upsilon_s^\alpha \mid \Upsilon_s^\alpha \in \mathbf{\Upsilon_s}\}$ are denoted by $\tau_{\mathbf{G_s}}$, $\tau_{\mathbf{T_s}}$, $\tau_{\mathbf{\Pi_s}}$, $\tau_{\mathbf{M_s}}$, $\tau_{\mathbf{\Gamma_s}}$ and $\tau_{\mathbf{\Upsilon_s}}$, respectively, such that the sets in $\mathbf{G_s}$, $\mathbf{T_s}$, $\mathbf{\Pi_s}$, $\mathbf{M_s}$, $\mathbf{\Gamma_s}$ and $\mathbf{\Upsilon_s}$ are distinct by definition. The αth elements of $\tau_{\mathbf{G_s}}$, $\tau_{\mathbf{T_s}}$, $\tau_{\mathbf{\Pi_s}}$, $\tau_{\mathbf{M_s}}$, $\tau_{\mathbf{\Gamma_s}}$ and $\tau_{\mathbf{\Upsilon_s}}$ are denoted by $\hat{G}_s^\alpha$, $\hat{T}_s^\alpha$, $\hat{\Pi}_s^\alpha$, $\hat{M}_s^\alpha$, $\hat{\Gamma}_s^\alpha$ and $\hat{\Upsilon}_s^\alpha$, respectively.

Finally, some mappings relating the primitives are defined.

Definition 3.3: *A Refining Mapping from a node s to its subnode t, ψ_t^s is a mapping from the space of tasks and constraints of node s to the space of tasks and constraints of node t, i.e.,*

$$\psi_t^s : \tau_{\mathbf{T_s}} \times \tau_{\mathbf{\Gamma_s}} \rightarrow \tau_{\mathbf{T_t}} \times \tau_{\mathbf{\Gamma_t}}, \qquad t \in \mathcal{I}_s^{[1]}.$$

Definition 3.4: *A Procedure Association Mapping at node s, P_s^s is a mapping from the space of tasks, constraints, measurements and resources to the space of procedures of the node s, i.e.,*

$$P_s^s : \tau_{\mathbf{T_s}} \times \tau_{\mathbf{\Gamma_s}} \times \tau_{\mathbf{M_s}} \times \tau_{\mathbf{\Upsilon_s}} \rightarrow \tau_{\mathbf{\Pi_s}}.$$

Definition 3.5: *A Task Accomplishment Function, $A^{s,n}$ is a function from the space of tasks and constraints of a node s and procedures, measurements and resources of its the (sub)nodes to $\{0,1\}$, i.e.,*

$$A^{s,n} : \tau_{\mathbf{T_s}} \times \tau_{\mathbf{\Gamma_s}} \times \{\tau_{\mathbf{\Pi_t}} \times \tau_{\mathbf{M_t}} \times \tau_{\mathbf{\Upsilon_t}}\}_{t\in\mathcal{I}_s^{[n]}} \rightarrow \{0,1\},$$

so that for a set $V_{s,t} \in \hat{T}_s^\alpha \times \tau_{\mathbf{\Gamma_s}} \times \{\hat{\Pi}_t^{\beta_t} \times \tau_{\mathbf{M_t}} \times \tau_{\mathbf{\Upsilon_t}}\}_{t\in\mathcal{I}_s^{[n]}}$,

$$A^{s,t}(V_{s,t}) = \begin{cases} 1\,; & \text{if } \hat{\Pi}_t^{\beta_t}\text{'s accomplish } \hat{T}_s^\alpha \text{ given other primitives,} \\ 0\,; & \text{otherwise,} \end{cases},$$

where n is either 0, or 1.

[3] In the earlier publications, these definitions are slightly different

Definition 3.6: *A Goal Decomposition Mapping at a node t, Δ_t^t is a mapping from the space of goals and constraints of node t to the space of tasks and constraints of the same node, i.e.,*

$$\Delta_t^t : \mathcal{T}_{\mathbf{G_t}} \times \mathcal{T}_{\mathbf{\Gamma_t}} \rightarrow \mathcal{T}_{\mathbf{T_t}} \times \mathcal{T}_{\mathbf{\Gamma_t}}.$$

Definition 3.7: *A Goal Formation Mapping from a node s to its subnode t, Φ_t^s is a mapping from the space of tasks and constraints of node s to the space of goals and constraints of subnode t, i.e.,*

$$\Phi_t^s : \mathcal{T}_{\mathbf{T_s}} \times \mathcal{T}_{\mathbf{\Gamma_s}} \rightarrow \mathcal{T}_{\mathbf{G_t}} \times \mathcal{T}_{\mathbf{\Gamma_t}}, \qquad t \in \mathcal{I}_s^{[1]}.$$

In the definitions of mappings, names like "refining", "procedure association", "goal decomposition" and "goal formation" are used. These names are for reference purposes only. A mapping is not required to perform the function that its name implies, just because it has named that way. Indeed, in the definitions, only the spaces of the mappings are described. If the only restrictions were the above definitions, any mapping with the matching domain and range would have been acceptable. So, some restrictive requirements should be imposed on these mappings, such that they are forced to function the way their names imply. The following definition imposes those requirements.

Definition 3.8: *A structurally coordinated hierarchy is said to be* functionally coordinated, *if there exists mappings and primitives, as described above, which satisfy the following* Functional Coordinability Axioms.

Functional Coordinability Axioms

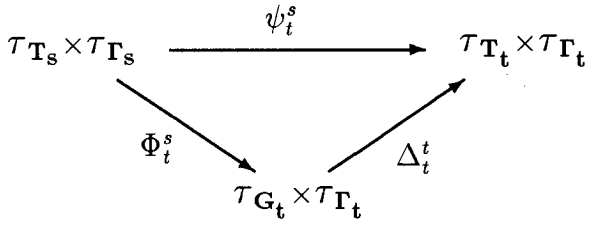
1. ψ_t^s *and* $\psi_t^{s^{-1}}$ *are both continuous.*
2. $\psi_t^s(\emptyset) = \emptyset$.
3. $\forall\, V_s \in \emptyset \times \mathcal{T}_{\mathbf{\Gamma_s}}, \quad \psi_t^s(V_s) \in \emptyset \times \mathcal{T}_{\mathbf{\Gamma_t}}$.
4. $\forall\, V_s^\alpha, V_s^\beta \in \mathcal{T}_{\mathbf{T_s}} \times \mathcal{T}_{\mathbf{\Gamma_s}}$,
$$\psi_t^s(V_s^\alpha \cup V_s^\beta) = \psi_t^s(V_s^\alpha) \cup \psi_t^s(V_s^\beta), \quad \forall t \in \mathcal{I}_s^{[1]}.$$
5. $\forall\, V_s^\alpha, V_s^\beta \in \mathcal{T}_{\mathbf{T_s}} \times \mathcal{T}_{\mathbf{\Gamma_s}}$,
$$\psi_t^s(V_s^\alpha \cap V_s^\beta) = \psi_t^s(V_s^\alpha) \cap \psi_t^s(V_s^\beta), \quad \forall t \in \mathcal{I}_s^{[1]}.$$
6. $\forall\, V_s \in \mathcal{T}_{\mathbf{T_s}} \times \mathcal{T}_{\mathbf{\Gamma_s}} : \quad \psi_t^s(V_s) = \emptyset \iff s \in \{\, u \mid \mathcal{I}_u^{[1]} = \emptyset \,\}$.
7. $\hat{T}_s^\alpha = \emptyset \iff P_s^s(\hat{T}_s^\alpha \times V_s) = \emptyset, \quad \forall\, V_s \in \mathcal{T}_{\mathbf{\Gamma_s}} \times \mathcal{T}_{\mathbf{M_s}} \times \mathcal{T}_{\mathbf{\Upsilon_s}}$.
8. $\forall\, \hat{\Pi}_s^\alpha \neq \emptyset,\ \forall\, V_s \in \mathcal{T}_{\mathbf{T_s}} \times \mathcal{T}_{\mathbf{\Gamma_s}} \times \mathcal{T}_{\mathbf{M_s}} \times \mathcal{T}_{\mathbf{\Upsilon_s}} : \quad \hat{\Pi}_s^\alpha \in P_s^s(V_s)$
$$\Longrightarrow A^{s,0}(\hat{\Pi}_s^\alpha \times V_s) = 1.$$
9. $\forall\, V_s \in \mathcal{T}_{\mathbf{\Gamma_s}} \times \mathcal{T}_{\mathbf{M_s}} \times \mathcal{T}_{\mathbf{\Upsilon_s}}$, *and* $\forall\, \hat{T}_s^\alpha \cap \hat{T}_s^\gamma \neq \emptyset$;
$$P_s^s\left((\hat{T}_s^\alpha \cup \hat{T}_s^\beta) \times V_s\right) \subseteq P_s^s\left(\hat{T}_s^\alpha \times V_s\right)$$

$$\subseteq P_s^s\left((\hat{T}_s^\alpha \cap \hat{T}_s^\gamma)\times V_s\right).$$

10. $\forall V_s \in \mathcal{T}_{\mathbf{T_s}}\times\mathcal{T}_{\mathbf{\Gamma_s}}$, *and* $\forall t \in \mathcal{I}_s^{[1]}$: $\psi_t^s(V_s) \neq \emptyset$
$\Longleftrightarrow \exists U_t \in \mathcal{T}_{\mathbf{M_t}}\times\mathcal{T}_{\mathbf{\Upsilon_t}}$: $\hat{\Pi}_t^\alpha \in P_t(\psi_t^s(V_s)\times U_t)$,
and $A^{s,1}(V_s\times\{\hat{\Pi}_t^\alpha\times U_t\}_{t\in\mathcal{I}_s^{[1]}}) = 1$.

11. $\Delta_0^0 \equiv$ *Identity mapping.*

12. *The following diagram commutes.*



The functional coordinability axioms provide restrictions to shape the relations of primitives among themselves. Most of the axioms express the notions described earlier mathematically.

The first six axioms define some properties of the refining mapping. Here is a partial list.

1. Neighborhoods of task-constraint pairs are mapped to neighborhoods of task-constraint pairs of the subnodes[4].
2. No task-constraint pairs are refined from an empty task-constraint pair.
3. Unions and intersections of task-constraint pairs can be taken before or after refinement.
4. Only the tip nodes have no further refinement.

The refining mapping is well behaved for the unions and intersections of task-constraint pairs. As an example, consider two task-constraint pairs, which adjust the angle of a robotic joint under different constraints. The forth and the fifth axioms guarantee the existence of a third task-constraint pair, which changes the angle under both of the constraints, gets refined to the combination of the refinements of the first two pairs.

The restrictions on neighborhoods of task-constraint pairs are especially important for different instances of a task. For example, assume two instances of a task, which change an angle of a robotic joint by 20° and 40°. Assume also that they are refined to subnode tasks which apply 5 volts to the joint input for 1 second and 2 seconds, respectively. Then, it is desirable to refine an angle changing task into another instance of the voltage applying task. However,

[4] Although the statement about mappings of neighborhoods is mathematically incorrect, it is the closest informal description of this property.

this desired behavior is only local. Extension requires more structure on the space of tasks and constraints than defined here.

The refining process terminates at the tip nodes, not before. This requirement in the sixth axiom intends to force the relations of the primitives to extend the full depth of the hierarchy. Although, mathematically, it is possible to have identity mappings instead of refinements, this should be avoided in practice. The next four axioms restrict the relations among procedures and task-constraint pairs.

The procedure association mapping is restricted in the following manner.

1. It associates at least one procedure for each task and one task for each procedure.
2. Any procedure associated with a task accomplishes the task.
3. Any procedure which accomplishes two tasks at the same time, also accomplishes each of the tasks.
4. Any procedure which accomplishes a task, also accomplishes any portion of the task.
5. For every refinement of a task-constraint pair, there exists a procedure, and its accomplishment is foreseen from the refining node.
6. If the accomplishment of a task-constraint pair can not be foreseen, then there should be no further refining.

Some of these restrictions are for good book-keeping, such as requiring a procedure to be associated with a task. Some restrictions require the inclusion of procedures with related tasks. And some others enforce the knowledge about the capabilities of the subnodes to be complete.

The last two axioms relate goal primitives to the others. Since refining mappings refine tasks of a node to tasks of its subnodes, goal formation and decomposition should be consistent with this refinement process. Finally, to start up the refinement, the goal decomposition at the top level is assumed to be trivial.

The most important result for a structurally and functionally coordinated system is in the following theorem.

Theorem 3.4: *For any given* $G_0^\alpha \in \mathcal{T}_{\mathbf{G_0}}$, *at every node* s, *there exists some* $V_s \in \mathcal{T}_{\mathbf{M_s}} \times \mathcal{T}_{\mathbf{\Gamma_s}} \times \mathcal{T}_{\mathbf{\Upsilon_s}}$, *such that* G_0^α *can be accomplished by local procedures in a structurally and functionally coordinated system.*

The proof of this theorem is also included in the appendix.

The next step is to describe the control flow associated with a structurally and functionally coordinated system.

4. CONTROL PROCESS

In this section, a control process based on the functionality assignments described in the previous section will be defined. The process will require a structurally and functionally coordinated system. Here, the control process

will be assumed identical at every node to enable a uniform treatment. However, as long as input-output relations of nodes are preserved, any type of control process including conventional, Petri net, fuzzy logic and neural network controllers can be utilized.

4.1. Describing the Control Flow

The only primitives transmitted down in the hierarchy are goals, constraints and resources. From a broader point of view, all the other primitives are merely internal values. To be able to process these transmitted primitives, the first step is to decompose them into task-constraint pairs. The goal decomposition mapping described above are for this purpose, and it incorporates the fifth element, *task decomposition* of the model mentioned in [15]. Part of this step is also to determine the applicable procedures under the limitations of constraints, resources and available measurements. Here, it is possible to obtain more than one applicable procedure set. As a result, a selection process is needed. In the next step, this selection is performed according to some criteria. Once a selection has been made, local procedures and subprocedures are obtained. Local procedures are applied as soon as their constraints are satisfied. Subprocedures have to go through another process to form new goals for the subnodes. This process is represented above by the goal formation mapping. At this final step, nodes form goals, constraints and resources for the subnodes, such that accomplishments of all the goals of the subnodes imply the accomplishment of the initial goal of the node.

Before, more details about the process is given, a schematic representation of the control process is given in Figure 4.5. The process is identical at every node, and in the figure the process is shown for a single node only. The *stiffness* and *cost* values attached to the transmitted primitives will be explained later.

The knowledge at a node is divided into two types. The first type consists of the knowledge about the capabilities of the node itself and its subnodes. This knowledge is given to the structure a priori in terms of the primitives, such as tasks, procedures and constraints, and their relations with each other, such as associating procedures with tasks. The completeness of this knowledge is very important to cover almost all the functionalities of a given system. Details of this type have been given in the previous section.

The second type of knowledge consists of the knowledge which utilizes the first type. The goal decomposition, selection of a procedure set and the goal formation are as a result of the second type of knowledge.

4.2. Formalizing the Control Flow

The knowledge to decompose goals into task-constraint pairs may become very complicated. Incorporating some sophisticated reasoning process is one possibility. On the other hand, it can be as simple as using look-up tables. Perhaps the most efficient approach is to use more elaborate methods at the higher levels of the hierarchy, and simpler methods at the lower levels. A somewhat

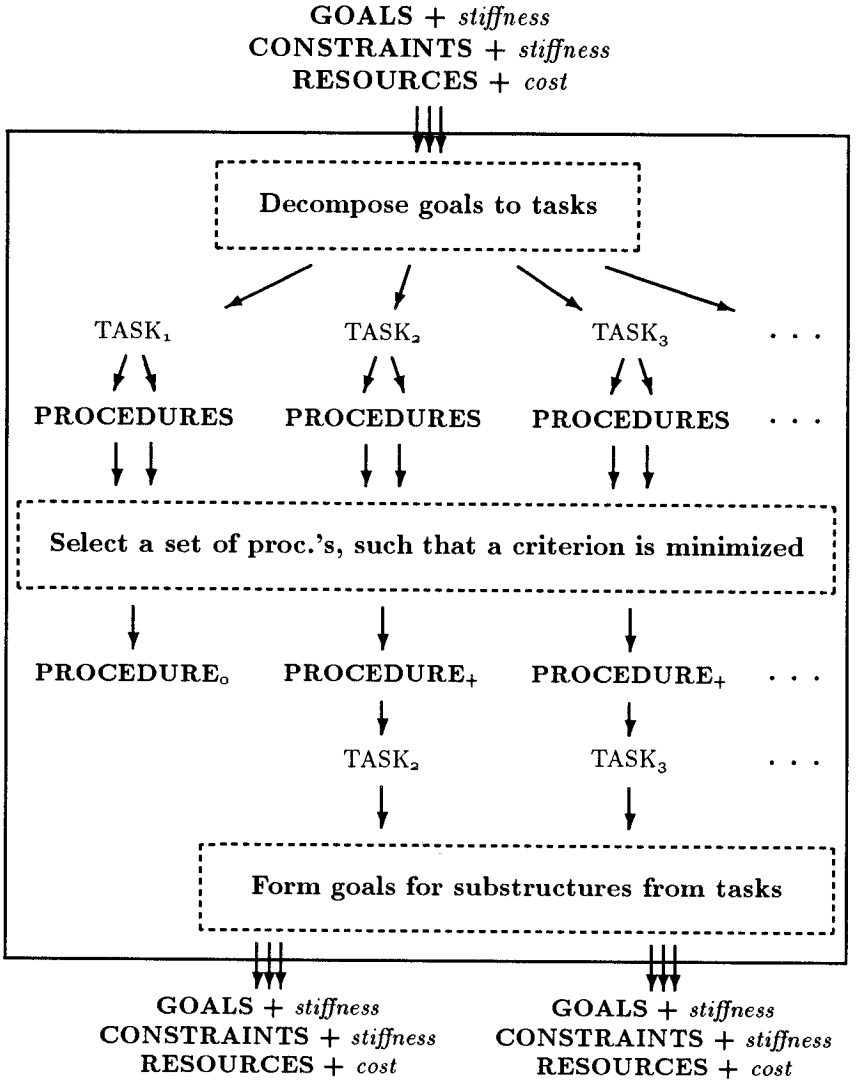


Figure 4.5: Control process diagram for each node.

different approach is needed to form the goals for the subnodes. The subprocedures, which are selected to accomplish the tasks, actually contain information about the subnodes' tasks. Therefore, most of the reasoning has been done, before the goal formation process starts.

The selection of procedures among a large number candidates requires a stage of optimization. Generally, a decentralized cost criterion is minimized or maximized, [17,18]. This function can be total energy, entropy or a combination of physical quantities, provided that an extremum exists. It can even be the sixth element, *value judgment* of the model mentioned in [15]. However, in order to have a reasonable selection, the importance and the priorities of certain primitives should be specified. For example, a constraint on the maximum allowable grasping tension at the end-effector of a robotic manipulator can be relaxed, if the manipulator is handling a metal block. On the other hand, the constraint has to be observed strictly, if it is handling an egg. In the selection process discussed here, a criterion will be assumed to be provided, such that a minimum exists.

To represent the strictnesses of goals, tasks and constraints, a set of *stiffness constants*, $\zeta_{G_s^i}$, $\zeta_{T_s^i}$ and $\zeta_{\Gamma_s^i}$ are defined for the initial primitives G_s^i, T_s^i and Γ_s^i, respectively, such that $\zeta \in \{0\} \cup \Re^+ \cup \{\infty\}$, where $\Re^+$ is the positive real numbers. For essential primitives, $\zeta = \infty$, and strictness decreases as $\zeta \to 0$. When relative orders of primitives are not so obvious, a finite number of values or even fuzzy qualifications may be used for the ζ's.

The initial resources and measurements have to be associated with another kind of constant to represent the cost of using them. These are called *cost constants*, and they are denoted by $\eta_{\Upsilon_s^i}$ and $\eta_{M_s^i}$ for resources and measurements, respectively[5]. Again, real values are assumed for these constants, such that $\eta \in \{-\infty\} \cup \Re \cup \{\infty\}$, where $\Re$ represents the real numbers. Here, $\eta = 0$ represents no gain or loss, $\eta > 0$ a decrease in resources or a penalty for using measurements, and $\eta < 0$ an increase in resources or an encouragement to use measurements.

The introduction of these constants may help selection within a node, but they need to be incorporated with the formal definitions. The stiffness and cost constants are initially assigned only to the initial sets of the primitives. These definitions should be somehow extended to the other elements of the topologies. The following definition provides these extensions.

Definition 4.9: *Given the stiffness and cost constants for the initial sets; the stiffness constants for goals, tasks and constraints are defined by*

$$\zeta_{U_s} = \sup_{V_s} \{\zeta_{V_s} \chi_{U_s}(V_s)\},$$

[5] In earlier publications, different versions of stiffness and cost constants were referred by α and β, respectively. However, since α and β are used for the elements in various topologies in this chapter, and the definitions differ slightly, new variables are introduced to prevent any possible confusion.

for all U_s *in* $\tau_{\mathbf{G_S}}$ *or* $\tau_{\mathbf{T_S}}$ *or* $\tau_{\mathbf{\Gamma_S}}$, *and* V_s *in* $\mathbf{G_S}$ *or* $\mathbf{T_S}$ *or* $\mathbf{\Gamma_S}$, *respectively, where*

$$\chi_U(V) = \begin{cases} 1 & \text{if } U \cap V \neq \emptyset \\ -\infty & \text{if } U \cap V = \emptyset \end{cases}.$$

The cost constants for resources and measurements are defined by

$$\eta_{U_s} = \sup_{V_s} \{\eta_{V_s} \chi_{U_s}(V_s)\},$$

for all U_s *in* $\tau_{\mathbf{M_S}}$ *or* $\tau_{\mathbf{\Upsilon_S}}$, *and* V_s *in* $\mathbf{M_S}$ *or* $\mathbf{\Upsilon_S}$, *respectively. The stiffness constant pairs in the product space* $\tau_{\mathbf{T_S}} \times \tau_{\mathbf{\Gamma_S}}$ *are defined by*

$$\zeta_{U_s} = \Big(\sup_{V_s, W_s} \{\zeta_{V_s} \chi_{U_s}(V_s \times W_s)\}, \sup_{V_s, W_s} \{\zeta_{W_s} \chi_{U_s}(V_s \times W_s)\} \Big),$$

for all U_s *in* $\tau_{\mathbf{T_S}} \times \tau_{\mathbf{\Gamma_S}}$, V_s *in* $\mathbf{T_S}$, *and* W_s *in* $\mathbf{\Gamma_S}$.

4.3. Obtaining Consistent Local Decisions

Another important consideration is to obtain a consistent propagation of the stiffness and cost constants. To achieve this consistency, some restrictions on these constants have to be imposed. The requirements for this consistency are described in the next definition.

Definition 4.10: *A structurally and functionally coordinated hierarchy is said to be* functionally consistent, *if it satisfies the following* Functional Consistency Axioms.

Functional Consistency Axioms

1. $\forall t \in \mathcal{I}_s^{[1]}: \quad \zeta_{\psi_t^s(\hat{T}_s^\alpha \times \hat{\Gamma}_s^\beta)} \geq \zeta_{\hat{T}_s^\alpha \times \hat{\Gamma}_s^\beta} = \sup_{u \in \mathcal{I}_s^{[-1]}} \zeta_{\psi_s^{u^{-1}}(\hat{T}_s^\alpha \times \hat{\Gamma}_s^\beta)}.$
2. $\forall t, s: \hat{M}_s^\alpha \cap \hat{M}_t^\beta = \hat{M}_t^\beta \Longrightarrow \eta_{\hat{M}_s^\alpha} \geq \eta_{\hat{M}_t^\beta}.$
3. $\forall t, s: \hat{\Upsilon}_s^\alpha \cap \hat{\Upsilon}_t^\beta = \hat{\Upsilon}_t^\beta \Longrightarrow \eta_{\hat{\Upsilon}_s^\alpha} \geq \eta_{\hat{\Upsilon}_t^\beta}.$

The functional consistency axioms are only guidelines to check the assignments of the stiffness and cost constants. In order to completely make the hierarchy optimize consistently, more restrictions should be imposed on the individual minimization criteria. Since our intent is to be flexible, no more restrictions will be imposed here.

In a functionally consistent system, the task-constraint pairs obtained from the task refining mappings must have equally or more strict stiffness constants. Moreover, the task-constraint pairs which may be refined from different supernodes must be assigned the most strict stiffness constants. These two requirements are from the the first functional consistency axiom. The second and the

third axioms consider the cases where a measurement or a resource is utilized in more than one node.

Up to this point, a nominal control process is described. This process completely depends on the correctness and the completeness of the knowledge incorporated. The planning portion may lack the consideration of uncertainities and unpredictable changes. To remedy this situation, practical issues in local and global feedback methods will be considered in the next sections. Formal descriptions of these methods are rather complicated, and they will not be included here.

5. LOCAL FEEDBACK METHODS

Local feedback methods consist of control strategies which utilize available measurements in real time. Local methods are confined within each node, and excluding measurements, there is no cooperation or information exchange among nodes. Because of all the desirable properties of feedback, the type of procedures, which utilize measurements, should be preferred whenever possible.

Some systems come with adequate measurements to apply feedback controllers, some do not. For example, the fischertechnik® robot arm has semi-accurate measurements of joint angles. On the other hand, its end effector has no pressure sensor, so even a fragile object has to be grasped without feedback. For the occasions when feedback is not possible, there is always a higher risk of failure. As a result, some other issues, such as failure detection and handling become important. These issues will be discussed later.

Some local procedures, which implement feedback, frequently use measurements from other nodes of the hierarchy. Whenever such a measurement is used, a smart link between the nodes should be established. This link would be temporary, and it would be disconnected by the node requiring the measurement. The link should also be smart enough to seek, interpret and relate the information at a detail meaningful to the node. The location of an object to be picked up by a robotic manipulator is a good example. The top node or any other high level node may need to know the location of the object in relation to the end effector. However, the exact coordinates involve too much detail for that node. So, a vicinity or a general direction would be desirable, and the link is expected to provide it.

A structurally coordinated system forms a hierarchy by definition, but inclusion of smart links may induce temporary loops into the system. The loops, on the other hand, do not introduce cyclic delegation of tasks during planning, since the links only transfer measurements.

6. GLOBAL FEEDBACK METHODS

Global feedback methods consist of strategies that adjust the planning methods according to the current status of the plan. This type of feedback involves

information exchange among nodes, i.e., they are "inter-node" feedback control strategies.

Similar to classical feedback methods, the global feedback process observes the output, and it supplies the necessary changes in planning to eliminate the unplanned behavior. On the other hand, unlike the classical feedback methods, the planned behavior and the observed output do not belong to the same class of representations. These different representations necessitate a tip-up information flow propagating node by node up in the hierarchy, [19,20]. Due to the nature of feedback not only the successes, but also the failures should be reported. Therefore, a careful classification of feedback reports and responses to these reports should be prepared.

6.1. Representing Feedback Information

To span all possibilities, four types of reports, which represents the global feedback information, are introduced as in [21].

Completion Reports inform a supernode the accomplishment of the goals originated from that node. They include an optional time stamp, and a list of procedures which were used to accomplish that goal.

Inclusion Reports are utilized both to initially transmit goals to lower level nodes and to inform the changes to connecting nodes. These reports add new goals, constraints or resources, and they also update the existing constraints or resources. If the reported constraints or resources already exist, the new values are set, and the change is assumed to be restricting for the constraints or relaxing for the resources.

Exclusion Reports remove the existing goals, constraints or resources due to the changing circumstances. The meaning of the reports for existing constraints or resources is the opposite of the inclusion report.

Failure Reports inform the higher level nodes about the failures of goals and briefly explain the bases of these failures. These reports are issued after an initial unsuccessful local replanning was performed. They contain a status field which describes the time element of the failure. For an expected failure in the future, the status may be a general information, such as "not-urgent" or a specific time duration, such as "10 hours".

Under certain circumstances, a constraint or a resource may be both restricting and relaxing. For these circumstances, both inclusion and exclusion reports are issued with the relevant portions included in these reports.

6.2. Detecting Changes

In order to submit one of these reports, detection methods need to be developed. Since detection methods depend on the type of the changes, they will be classified into two types.

Local Changes are the type of changes which are encountered during real-time execution. The effects of this type of changes are usually immediate and detections are made by the relevant nodes. Moreover, the detecting node decides, if a change is positive or negative. An example of such a change is the reduction of the maximum angular speed of one of the arms of a manipulator due to increased friction.

Remote Changes are the changes which are detected by nodes other than the relevant nodes. Their effects are usually not immediate, but the relevant nodes should be informed as soon as possible. The simplest way to identify the relevant nodes is to put marks on the constraints and resources utilized, during the initial planning process. Later, when the changes are detected, they are propagated first all the way to the top node, then throughout the hierarchy, so that the relevant nodes may be identified from their marks. (Here, an implementation of a black board architecture might improve the speed of response considerably.) An example of this type of a change is a decrease in temperature which might adversely effect the viscousity, thus the speed, of an hydraulic motor. The effects of this change might not be detected locally at the motor node, due to the large time-constant of the heat transfer.

6.3. Deciding on a Feedback Strategy

In a hierarchical organization, adjustment to planning methods may also be accomplished in various ways.

First order replanning is when replanning is performed under the assumption that the hierarchy stays the same.

Second order planning is when replanning techniques include some organizational changes to accommodate a better control strategy, but the new organization is still hierarchical.

Third order planning is when replanning techniques consider other organizations, such as dynamical and non-hierarchical for the "best" control.

The first order planning is not only the easiest, but it also has the ability to replan locally whenever possible.

In the optimal case, under any type of change, the replanning process starts from the top node and proceeds similar to the initial planning process. This method requires extensive time and work. Moreover, it is also impractical, if changes occur at irregular intervals during the execution. To eliminate this type of replanning, a strategy which uses the initial set of tasks may be utilized. This local replanning strategy is remedial, and it is suboptimal.

The replanning process at a node is triggered by an inclusion or an exclusion or a failure report. When a node either generates or receives such a report, it first decides on the character of the report. If the report is decreasing

the operation boundaries, it is accepted as a negative report, otherwise it is accepted as a positive report.

6.4. Replanning under Negative Changes

To consider replanning under negative changes, assume a node obtains a negative report. After obtaining the report, the node first decides if the report will lead to a failure in any of the primitives or not. This decision is based on the use of the changed variables by the selected procedures. If under the new conditions, the task decomposition process gives the same set of tasks, procedures and constraints, then the inclusion or exclusion report will not lead to a failure report. Otherwise, a failure report will be issued by that node.

Failure reports can be issued for all the primitives. However, failures in constraints, resources and measurements are the most common types of failures. Incorrect estimations, unpredictable events or malfunctions may all be reasons for these types of failures. Failures in procedures are usually issued as a result of failures in constraints, resources or measurements. But, these failures may also be due to incorrect or missing information in the knowledge base. Failures in tasks generally follow failures in procedures. Incorrect associations of procedures and incorrect decomposition rules may also lead to this type of failures. Finally, failures in goals are the most severe failures, and they are as a result of unresolved failures in tasks.

To observe the replanning process, we start with a negative inclusion or exclusion report at a specific node. This report might be generated at the node as a result of a local change or it can be received from another node as a result of a remote change. Irrespective of the source of the change, a negative report will cause either a failure in constraints, resources or measurements.

If a failure in constraints is issued, then the next step is to check the stiffness of the constraint. If the constraint has the highest level stiffness, i.e., the stiffness constant is infinity, then a failure in procedures will be declared. Otherwise, the new constraint will be reported to the relevant connecting nodes, and no other replanning action will be taken. For failures in resources and measurements, before checking the stiffnesses, the existence of an alternative source is checked. If such an alternative exists, then the alternative will be utilized. It will again be reported to the relevant connecting nodes. If there is no alternative, then the stiffnesses will be checked, and an identical action will be taken.

Failures in procedures are issued for three reasons.

1. Failures in constraints, resources and measurements are unresolved.
2. Measured results mismatch with the expected results.
3. A failure in goals is received from a subnode.

The first step after this type of a failure is to check for other procedures which will accomplish its associated task. Other choices will not give an optimal solution, but they will accomplish the task. If there is at least one more applicable procedure which will accomplish the associated task, then that procedure will be applied. If there is no such procedure, then the stiffness of the associated

task will be checked. If it has the highest level stiffness, then a "failure in task" will be declared. Otherwise, it will be reported to the supernodes, and normal execution will be continued.

Failures in tasks are issued for two reasons.

1. Failures in procedures are unresolved.
2. Completed tasks do not accomplish the decomposed goals.

Under these conditions, the goals, from which the failed tasks were obtained, will be decomposed again to get different sets of tasks. If this process is successful, then the new procedures will be checked. According to this check, the new procedures will either be applied and reported to the connecting nodes, or a failure in procedures will be issued. If this process is not successful, then a "failure in goals" will be issued.

Failures in goals are issued when failures in tasks are not resolved. The replanning for this type of a failure cannot be handled at this node, and it will be reported to the relevant supernodes.

Since higher level nodes have a more broader view of the planning process, some of the resolutions might not be acceptable by them. If this is the case, higher nodes can issue reports with an interrupt urgency status to get the attention of the lower level nodes.

During the replanning due to negative changes, there are two interrupt routines; one of them is soft, and the other one is hard. The soft interrupt is when new goals from supernodes are received. The receiving node first finishes up the current pattern and then goes back to decompose the new goals. The hard interrupt is when the time expires during replanning. If this happens, the node will stop all the processes, and it will run a retrieval plan. A retrieval plan is an alternative plan to reach an acceptable state, when the originally intended state is not achievable by any set of goals. These plans are usually pre-prepared for a large class of possible cases, and minor adjustments for the specific case are sufficient. The connecting nodes will also be informed about this decision.

6.5. Replanning under Positive Changes

The replanning under positive changes is simpler than the one due to negative changes. The replanning process tries to do better than initially intended. However, since the original procedures will accomplish the required goals, there is no real need to try harder for a new set of procedures. This approach definitely does not utilize the whole potential of the new situation.

When positive changes are measured or received, first the connecting nodes will be informed. Second, the nominal control process will be restarted this time with the new information. The starting point of this process will be either the decomposition of the goals or the selection of sets of tasks, procedures and constraints depending on the urgency of the replanning.

During the replanning due to positive changes, there is only one hard interrupt routine which is activated when response time expires. When this time

expires, the replanning process will stop, and the original set of procedures will be used.

7. CONCLUSIONS

In this chapter, a different hierarchical organization is introduced. The major difference of this organization is that it is based on the physical structure of a system, rather than its functionality. This object-oriented approach has some unique properties.

- Most of the functionality of a system can be utilized under the same organization.
- The hierarchical organization is not restricted to a tree hierarchy, where every node has at most one subnode.
- Decision making and controllers are distributed throughout the hierarchy for real-time parallel processing.
- Overlapping representations of the portions of a system can be incorporated, and as a result, many functionalities of these portions can be explored.
- A complete theoretical definitions of mathematically vague notions such as abstraction, summarizing and functional decomposition are easily obtained.
- The organization is modular in such a way that addition of new hardware does not alter the existing structure.
- Failures and replanning can be handled locally as much as possible.

It also has some additional desirable properties.

- The hierarchy is applicable to all functional systems.
- The design process naturally forms a hierarchy such that the system is described in increasing detail, and the intelligence decreases from top to tip.
- The system and its functionality are described multi-resolutionally and possibly redundantly at different detail to provide deep reasoning and planning

APPENDIX

In this appendix, the proofs of the theorems are presented.

Proof of Theorem 2.1: The proof has two parts.

1. $\Sigma_0 \supseteq \bigcup_{j:\mathcal{I}_j^{[1]}=\emptyset} \Sigma_j$.

2. $\Sigma_0 \not\supset \bigcup_{j:\mathcal{I}_j^{[1]}=\emptyset} \Sigma_j$.

The proof of the first part is trivial from the definition of the Borel set $\mathcal{B}_\Sigma$ in Σ_0, since the open set $\Sigma_j \subseteq \Sigma_0$ for all $\Sigma_j \in \mathcal{B}_\Sigma$. The second part is proven by contradiction. Assume $\Sigma_0 \supset \bigcup_{j:\mathcal{I}_j^{[1]}=\emptyset} \Sigma_j$. From the fourth structural coordinability axiom, $\Sigma_j \in S_{\max}$ for all j, such that $\mathcal{I}_j^{[1]} = \emptyset$, and $\mathcal{I}_j^{[0]} \neq \emptyset$, where $S_{\max}$ is as defined in the axiom. Let $\Sigma_u = \Sigma_0 \setminus \bigcup_{j:\mathcal{I}_j^{[1]}=\emptyset} \Sigma_j \in \mathcal{B}_\Sigma$. Then, from the definition of Σ_u, $\Sigma_u \neq \emptyset$, and $\Sigma_u \cap \Sigma_j = \emptyset$ for all j, such that $\mathcal{I}_j^{[1]} = \emptyset$. Next, define $S^*_{\max} = S_{\max} \cup \{\Sigma_u\}$. But $S^*_{\max} \in \mathcal{S}$, and $\overline{\overline{S}}^*_{\max} = \overline{\overline{S}}_{\max} + 1$ which contradicts the fourth axiom. □

Proof of Theorem 2.2: Let $S_{\max}$ be as in the fourth axiom. From this axiom, $\Sigma_j \in S_{\max}$ for all j, such that $\mathcal{I}_j^{[1]} = \emptyset$. Assume $\exists \Sigma_s \in \mathcal{B}_\Sigma$, and $\exists \Sigma_i \in S_{\max}$, such that $\Sigma_s \cap \Sigma_i \neq \emptyset$ and $\Sigma_s \cap \Sigma_i \neq \Sigma_i$. Let $\Sigma_u = \Sigma_s \cap \Sigma_i \in \mathcal{B}_\Sigma$ and $\Sigma_v = \Sigma_i \setminus \Sigma_s \in \mathcal{B}_\Sigma$. Then, $\Sigma_u, \Sigma_v \neq \emptyset$, $\Sigma_u \cup \Sigma_v = \Sigma_i$ and $\Sigma_u \cap \Sigma_v = \emptyset$. Now, let $S^*_{\max} = [S_{\max} \setminus \{\Sigma_i\}] \bigcup \{\Sigma_u, \Sigma_v\}$. But then $S^*_{\max} \in \mathcal{S}$ and $\overline{\overline{S}}^*_{\max} = \overline{\overline{S}}_{\max} + 1$ which contradicts with the fourth axiom. □

Proof of Theorem 2.3: Assume $\Sigma_s \cap \Sigma_t \neq \emptyset$ for a $t \notin \mathcal{I}_s^{[n]}$ for all integer n, and $\Sigma_s \cap \Sigma_t \not\subseteq \bigcup_{i=1}^{k} \Sigma_{m_i}$ for all k, m_1, p_1, q_1, ..., m_k, p_k, $q_k \in \mathcal{Z}^+$, such that $m_i \in \mathcal{I}_s^{[p_i]} \cap \mathcal{I}_t^{[q_i]}$ for all $i = 1, \ldots, k$. Let

$$\Sigma_u = \bigcup_{k \in \mathcal{Z}^+} \bigcup_{i=1}^{k} \bigcup_{p_i \in \mathcal{Z}^+} \bigcup_{q_i \in \mathcal{Z}^+} \bigcup_{m_i \in \mathcal{I}_s^{[p_i]} \cap \mathcal{I}_t^{[q_i]}} \Sigma_{m_i} \in \mathcal{B}_\Sigma,$$

and $\Sigma_v = \Sigma_s \cap \Sigma_t \setminus \Sigma_u$. Here, $\Sigma_v \neq \emptyset$, and $\Sigma_v \in \mathcal{B}_\Sigma$ as a consequence of the previous assumptions and definitions. From the previous theorem and the fourth axiom, $\exists \Sigma_w \in S_{\max}$, such that $\Sigma_v \cap \Sigma_w = \Sigma_w$. Therefore, from the definition of Σ_v, $\Sigma_w \subseteq \Sigma_s \cap \Sigma_t$. But then, $\Sigma_w \subseteq \Sigma_j$ for any $j \in \mathcal{I}_s^{[n_1]} \neq \emptyset$, and for any $j \in \mathcal{I}_t^{[n_2]} \neq \emptyset$, where n_1, $n_2 \in \mathcal{Z}^+$, as a result of repeated applications of the third structural coordinability axiom. Therefore, $\exists n_s \in \mathcal{Z}^+$, so that $w \in \mathcal{I}_s^{[n_s]}$, where $\mathcal{I}_s^{[n_s+1]} = \emptyset$, and $\exists n_t > 0$, such that $w \in \mathcal{I}_t^{[n_t]}$, where $\mathcal{I}_t^{[n_t+1]} = \emptyset$ which contradicts with the assumption. □

Proof of Theorem 3.4: To prove the theorem, the refining process will be first considered at one node. Then, the results will be extended for all the

nodes, since the process is identical at all the nodes. So, assume the nontrivial case, i.e., $G_0^\alpha \neq \emptyset$. Since, $\Delta_0^0 \equiv$ *Identity mapping*, $T_0^\alpha = G_0^\alpha$. Then, pick a set $V_0 \in \tau_{\mathbf{\Gamma_0}} \times \tau_{\mathbf{M_0}} \times \tau_{\mathbf{\Upsilon_0}}$, such that there exists $\hat{\Pi}_0^\beta \in P_0^0(T_0^\alpha \times V_0)$, and $\hat{\Pi}_0^\beta \neq \emptyset$. Such a procedure exists from the seventh axiom. Moreover, the procedure also accomplishes the task with the given primitives as a consequence of the eighth axiom, i.e., $A^{s,0}(T_0^\alpha \times \hat{\Pi}_0^\beta \times V_0) = 1$. At this point, if the procedure is a local procedure, i.e., $\psi_t^s\left(T_0^\alpha \times (V_0 \cap (\tau_{\mathbf{\Gamma_0}} \times \emptyset \times \emptyset))\right) = \emptyset$, then the proof is concluded. If not, then the tenth axiom guarantees that the accomplishment of all the refined tasks implies the accomplishment of the original task. Also from the twelfth axiom, the goal formation and goal decomposition mappings will obtain the same task refinement, i.e., $\Delta_t^t \circ \Phi_t^0\left(T_0^\alpha \times (V_0 \cap (\tau_{\mathbf{\Gamma_0}} \times \emptyset \times \emptyset))\right) = \psi_t^0\left(T_0^\alpha \times (V_0 \cap (\tau_{\mathbf{\Gamma_0}} \times \emptyset \times \emptyset))\right)$. Therefore, the initial goal is refined and distributed among the subnodes of the top node. As a result, the process is advanced one level down in the hierarchy. If all the selected procedures are local at a level before the tip, the proof is concluded at that level. If not, from axioms six, seven and eight, there exists local procedures which accomplish the refined tasks at the tip level. □

REFERENCES

[1] H. A. Simon, *The Sciences of the Artificial.* Cambridge, Massachusetts: The MIT Press, second ed., 1981.

[2] J. S. Albus, A. J. Barbera, and M. L. Fitzgerald, "Programming a hierarchical robot control system," in *Proceedings of the 6th International Robot Technology*, (Paris, France), pp. 505–517, June 1982.

[3] J. S. Albus, C. R. McLean, A. J. Barbera, and M. L. Fitzgerald, "Hierarchical control for robots and teleoperators," in *Proceedings of the IEEE Workshop on Intelligent Control*, (Troy, New York), pp. 39–49, August 1985.

[4] J. Rasmussen, "The role of hierarchical knowledge representation in decisionmaking and system management," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-15, pp. 234–243, March/April 1985.

[5] K. P. Valavanis, *A Mathematical Formulation for the Analytical Design of Intelligent Machines.* PhD thesis, Rensselaer Polytechnic Institute, Troy, New York, 1986.

[6] G. N. Saridis and K. P. Valavanis, "Software and hardware for intelligent robots," in *Proceedings of the Workshop on Space Telerobotics: Volume I*, (Pasadena, California), pp. 241–249, Jet Propulsion Laboratory, July 1987.

[7] A. M. Meystel, "Nested hierarchical controller with partial autonomy," in *Proceedings of the Workshop on Space Telerobotics: Volume I*, (Pasadena, California), pp. 251–270, Jet Propulsion Laboratory, July 1987.

[8] C. Işık and A. M. Meystel, "Pilot level of a hierarchical controller for an unmanned mobile robot," *IEEE Journal of Robotics and Automation*, vol. 4, pp. 241–255, June 1988.

[9] P. J. Antsaklis, K. M. Passino, and S. J. Wang, "Towards intelligent autonomous control systems: Architecture and fundamental issues," *Journal of Intelligence and Robotic Systems*, vol. 1, pp. 315–342, 1989.

[10] F.-Y. Wang and G. N. Saridis, "A coordination theory for intelligent machines," *Automatica*, vol. 26, pp. 833–844, September 1990.

[11] P. J. Antsaklis, K. M. Passino, and S. J. Wang, "An introduction to autonomous control systems," *IEEE Control Systems Magazine*, vol. 11, pp. 5–13, June 1991.

[12] K. P. Valavanis and H. M. Stellakis, "A general organizer model for robotic assemblies and intelligent robotic systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, pp. 302–317, March/April 1991.

[13] G. N. Saridis, "Analytic formulation of the principle of increasing precision with decreasing intelligence for intelligent machines," *Automatica*, vol. 25, pp. 461–467, May 1989.

[14] A. M. Meystel, "Nested hierarchical control," in *An Introduction to Intelligent and Autonomous Control* (P. J. Antsaklis and K. M. Passino, eds.), Norwel, Massachusetts: Kluwer Academic Publishers, May 1992.

[15] J. S. Albus, "A reference model architecture for intelligent system design," in *An Introduction to Intelligent and Autonomous Control* (P. J. Antsaklis and K. M. Passino, eds.), Norwel, Massachusetts: Kluwer Academic Publishers, May 1992.

[16] L. Acar and Ü. Özgüner, "Design of knowledge-rich hierarchical controllers for large functional systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, pp. 791–803, July/August 1990.

[17] L. Acar and Ü. Özgüner, "A completely decentralized suboptimal control strategy for moderately coupled interconnected systems," in *Proceedings of the 1988 American Control Conference*, (Atlanta, Georgia), pp. 1521–1525, June 1988.

[18] L. Acar and Ü. Özgüner, "A locally computed suboptimal control strategy for a class of interconnected systems," *Journal of Optimization Theory and Applications*, vol. 62, pp. 189–210, August 1989.

[19] D. C. Brown, *Expert Systems for Design Problem-Solving using Design Refinement with Plan Selection and Redesign.* PhD thesis, Department of Computer and Information Science, The Ohio State University, Columbus, Ohio, 1984.

[20] D. C. Brown and B. Chandrasekaran, "Knowledge and control for a mechanical design expert system," *IEEE Computer Magazine*, vol. 19, pp. 92–100, July 1986.

[21] L. Acar and J. R. Josephson, "Global feedback methods of a hierarchical controller for real-time execution and replanning," in *Proceedings of the Fifth IEEE International Symposium on Intelligent Control*, (Philadelphia, Pennsylvania), pp. 40–44, September 1990.