NISTIR 4888

# NIST SUPPORT TO THE NEXT GENERATION CONTROLLER PROGRAM: 1991 FINAL TECHNICAL REPORT

**James S. Albus**
**Richard Quintero**
**Frederick Proctor**
**John Michaloski**
**Nicholas Dagalakis**
**Nicholas Tarnoff**

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Robot Systems Division
Bldg. 220 Rm. B124
Gaithersburg, MD 20899

**Thomas Kramer**

Research Associate
Department of Mechanical Engineering
The Catholic University of America
Washington, DC 20064

NIST

NISTIR 4888

# NIST SUPPORT TO THE NEXT GENERATION CONTROLLER PROGRAM: 1991 FINAL TECHNICAL REPORT

**James S. Albus**
**Richard Quintero**
**Frederick Proctor**
**John Michaloski**
**Nicholas Dagalakis**
**Nicholas Tarnoff**

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards
and Technology
Robot Systems Division
Bldg. 220 Rm. B124
Gaithersburg, MD 20899

**Thomas Kramer**

Research Associate
Department of Mechanical Engineering
The Catholic University of America
Washington, DC 20064

July 1992


**U.S. DEPARTMENT OF COMMERCE**
**Barbara Hackman Franklin, Secretary**

**TECHNOLOGY ADMINISTRATION**
**Robert M. White, Under Secretary for Technology**

**NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY**
**John W. Lyons, Director**

TABLE OF CONTENTS

LIST OF FIGURES

# APPENDICES

## Product Endorsement Disclaimer

Reference to specific brands, equipment, or trade names in this document are made to facilitate understanding and do not imply endorsement by the National Institute of Standards and Technology.

# 1 REPORTING PERIOD

This report covers work performed by National Institute of Standards and Technology (NIST) Robot Systems Division between October 1, 1990 and December 31, 1991 in continuation of work previously undertaken by NIST during FY90 for the U.S. Air Force Next Generation Controller Program (NGC). This project is being conducted under AGREEMENT No. F33615-89-C-5706 and MIPR No. FY1133-91-N5056, dated 19 June 1991. The period of performance for this MIPR is from March 1, 1991 through September 30, 1991. Air Force funding for this effort was actually received on July 9, 1991.

The work reported here is being leveraged with other Robot Systems Division projects and is being partially funded using NIST internal funds as well as other agency funds (i.e., Navy MANTECH funds for the Automated Manufacturing Research Facility (AMRF)).

# 2 PERSONNEL ASSIGNED

During this reporting period, the following personnel participated in this effort:

NIST, Manufacturing Engineering Laboratory, Robot Systems Division—823

| | |
|---|---|
| Dr. James Albus | Principal Investigator |
| Richard Quintero | Deputy Project Manager |
| Fred Proctor | Project Technical Leader |
| Dr. Nick Dagalakis | ADACS |
| Hui-Min Huang | RCS Methodology |
| Adam Jacoff | ADACS |
| Roger Kilmer | ADACS/RCS Methodology |
| Dr. Tom Kramer | PDES/Enhanced Machine Tool Controller |
| John Michaloski | RCS Methodology |
| Rick Norcross | ADACS/RCS Methodology |
| Bob Russell | ADACS |
| Keith Stouffer | ADACS |
| Nicholas Tarnoff | ADACS/RCS Methodology |
| Tom Wheatley | RCS Methodology |

NIST, Manufacturing Engineering Laboratory, Automated Production Technology Division—822

| | |
|---|---|
| Don Blomquist | Enhanced Machine Tool Controller |
| Alkan Donmez | Enhanced Machine Tool Controller |

The appellation following each name indicates the principal area of that person's contribution:

ADACS (Advanced Deburring and Chamfering System) is the NIST robotics testbed for NGC. The ADACS project is also funded by Navy MANTECH funds for the Automated Manufacturing Research Facility (AMRF). The Navy money is paying for the robot and control system hardware, the computer controlled deburring tool, and the analysis of the deburring process. Air Force NGC funds have been proposed to support the development and installation of an NGC controller and NGC software on the ADACS workstation. During 1991, ADACS was used as a testbed for developing NML recommendations to the NGC team.

RCS Methodology is a NIST funded project that is using the Real-time Control System (RCS) reference model architecture to develop an engineering methodology and software tools for design and implementation of real-time intelligent control systems.

PDES indicates work on the problem of interfacing data to the NGC ADACS controller through the PDES (Product Data Exchange Specification) format.

Enhanced Machine Tool Controller indicates a NIST FY 92 project to bring up an NGC controller on a Monarch vertical spindle mill for a machining workstation in the AMRF. This project will also be funded by Navy MANTECH funds for the AMRF. The Navy money will pay for the machine tool, control system hardware, and studies of accuracy enhancement through software thermal compensation and tool wear compensation. The proposed FY 92 Air Force NGC funds for Tasks 4 and 5 will support the development and installation of an NGC controller and NGC software on the machining workstation.

# 3 PROGRESS

## 3.1 Task 1: Review and Critique NGC Technical Activities/Documents

This year's effort started with a "kickoff" meeting between Martin Marietta (MM) and NIST on October 26, 1990 at NIST in Gaithersburg, Maryland. Following that meeting NIST personnel participated in NGC Industry Review Board (IRB) meetings on December 4-5, 1990 and June 25-26, 1991.

At the December 4–5 1990 IRB meeting, Jim Albus gave a presentation on the NIST work on a Strawman NML and Strawman World Model for the NGC.

At the June 25–26 1991 IRB meeting, Jim Albus presented the NIST work on Task Frames, Command Frames, and World Model information needed to support the NML.

Jim Albus, Fred Proctor, John Michaloski and Tom Wheatley participated in the NGC IDEF Module Decomposition Workshop in Denver on February 5–7, 1991. As a result of that meeting, NIST personnel generated a mapping between the NGC IDEF diagrams reviewed at the workshop and the NIST NASREM architecture. This technical report was forwarded to MM as a memo on February 27, 1991.

NIST hosted an FY92 planning meeting with Mickey Hitchcock and Martin Marietta, on October 16, 1991, at NIST in Gaithersburg, Maryland. During that meet-

ing we discussed how NIST might change its role, in support of the NGC program, from primarily a government consultant to one of active support for the NGC Demonstration Tasks 4 through 7. We also discussed the possibility of a NIST role in the Low End Controller (LEC) project.

NIST personnel, Jim Albus, Fred Proctor and John Michaloski, attended the NGC TAG meeting on October 17 and 18, 1991, in Denver, Colorado. The purpose of this meeting was to hold a technical discussion of the progress Martin Marietta and its subcontractors have made toward the SOSAS. The current state of the SOSAS was presented and the meeting participants reviewed the technical specification and design of the NGC.

Jim Albus, Rick Quintero, John Michaloski, and Alkan Donmez participated in the NGC/LEC Control Builders Workshop at MMC, Littleton, Colorado starting on October 28, 1991 through November 11, 1991. Jim Albus made a presentation at that meeting which reviewed our current thinking in the areas of NML, World Model data structures and performance measures for robots.

Tom Kramer reviewed a draft University of Texas paper on NGC information models, the first draft of schemas for NML written in EXPRESS.

## 3.2 Task 2: Develop Strawman NML Commands and Protocols

In order to accomplish this task, it is necessary to define the functionality of the NGC modules. For example, what needs to be communicated between modules depends on what the modules already know. If they know everything, then nothing needs to be communicated. If they know nothing, then everything needs to be communicated. In the NGC, it is assumed that the modules know quite a bit. What they know, and how that knowledge is stored and used is critical to the performance of NGC in general, and to the structure of the NML in particular.

Throughout the past year, there has been a great deal of discussion and controversy within the NGC project as to the definition of NGC modules, particularly related to the granularity and functionality of those modules. Some have argued for course granularity with the internal structure of the NGC subsystems left up to the vendors. Others have argued for fine granularity so that the internal structure of the subsystems is visible to the users in a truly "open system architecture".

As a result of this controversy, the definition of the NGC modules has been in a state of flux, with a great deal of uncertainty as to what the final form of the NGC Specification for an Open System Architecture Standard (SOSAS) will be. One approach to this problem has been to attempt to define SOSAS conformance classes. From this approach came the Low End Controller, which represents the coarsest of the proposed granularities, with the least functionality.

NIST has taken the opposite approach. NIST has chosen to pursue the finest possible granularity, with the most functionality. Our rational is that, given a sufficiently fine granularity, any of the proposed NGC systems can easily be achieved by simply clustering fine granularity modules into the desired coarser granularity modules. Adding fine granularity modules can increase functionality, and hence, any of the NGC conformance classes can be achieved from a single set of fine granularity building blocks.

In pursuing this approach, NIST has used the RCS (Real-time Control System)

reference model architecture that has been developed at NIST over the past 15 years. RCS has a systematic regularity and recursive structure that has recently been developed into a canonical form for intelligent machine systems. RCS is perhaps best known as a precursor to the NBS/NASA Reference Model architecture (NASREM) for telerobotic systems. Partly because NASREM influenced both the requirements and the system architecture of the NGC, there exists a close correspondence between the RCS reference model and the NGC architecture. A description of that correspondence is given below.

Translation from the RCS reference model representation into the NGC architecture as defined in the June 1991 Next Generation Workstation/Machine Controller Architecture Definition Document is thus straightforward. The translation from RCS to NGC has been carried out in several sections of this report without difficulty. The remainder of the translation will be carried out during the coming year's activities. It should also be noted that NIST is currently working on representing portions of the RCS reference model in EXPRESS, using ALPS to describe process plans, and using PDES/STEP for part descriptions. This will further facilitate easy translation between RCS and NGC SOSAS representations.

Among the advantages of the RCS→NGC approach is that an enormous amount of prior and on-going work on RCS can be leveraged into the NGC program. For example, much of the work in the NIST Automated Manufacturing Research Facility and particularly the current work on robotic deburring under the Advanced Chamfering and Deburring System (ADACS) project, as well as the work on Enhanced Machine Tool Control can be immediately applied to the NGC program. In addition, much of the work done by NIST for NASA on space station telerobotics, for DARPA on intelligent control systems for undersea vehicles and nuclear submarines, the work being done for the Bureau of Mines on coal mine automation, the work being done for the Army/Marine Corps unmanned ground vehicle program, the work being done for the U.S. Postal Service on Post Office automation, and the work internally funded under the Department of Commerce Intelligent Machine Systems initiative can be brought to bear on the NGC problem domain. For example, during the past two years under these programs a theoretical model for intelligent systems has been developed, RCS has evolved into a canonical form, and a variety of intelligent system design techniques, task analysis methodologies, and software development tools have been developed. These are summarized in two recent publications, "Outline for a Theory of Intelligence", and "A Reference Model Architecture for Intelligent Systems Design". Copies of these documents are contained in Appendix A. A methodology for developing hierarchical controllers, the "Handbook for Real-Time Intelligent System Design" has also been written, and is to be published.

### 3.2.1 A Mapping Between RCS and NGC

The following is an example of a mapping between RCS and NGC, which is based on the June 1991 version of the NGC Architecture Definition Document prepared by Martin Marietta:

*The NGC Workstation Management Subsystem*

The NGC Workstation Executive Module corresponds to the RCS level 5 (workstation) Job Assignment submodule.

NGC External Coordinator Module input and data distribution functions are handled in RCS by the level 5 (workstation) Job Assignment submodule. Output functions of the NGC External Coordinator Module are handled in RCS by the level 5 Executor submodules.

NGC configuration Manager Module and Asset Manager Module functions are also performed by the RCS level 5 Job Assignment submodule.

NGC Health Manager Module functions are handled at the appropriate levels by RCS Executor submodules at all levels of the RCS hierarchy. At each level, each RCS Executor submodule monitors the health and safety of the subsystem under its control and reports problems through status reports posted in the world model and/or transmitted to the level above.

The NGC Workstation Planning Subsystem corresponds directly to the RCS planner submodules at levels 3, 4, and 5. These generate Control Plans, Coordinated Control Plans, and Operations Plans respectively. The principle difference between NGC and RCS is that the NGC Workstation Planning Subsystem operates off-line, whereas the RCS planners at each level operate under the real-time constraints appropriate to their respective level.

In particular, the NGC Operations Planner Module corresponds directly to the set of RCS level 5 (workstation) Planners that generate plans for each of the pieces of equipment in the workstation.

The NGC Task Planner Module corresponds directly to the set of RCS level 4 (task) Planners that generate and coordinate sequences of NGC task elements (RCS E-Moves) for each of the equipment subsystems.

The NGC Path Planner Module corresponds directly to the set of RCS level 3 (E-move) Planners that generate and coordinate sequences of collision-free paths for tools, inspection probes, or grippers.

The NGC Modeler Module corresponds directly to the set of models that reside in the RCS world model at their respective levels.

*The NGC Human Interface Subsystem*

The NGC Human Interface Subsystem corresponds exactly to the RCS operator interface module.

*The NGC Task Execution Subsystem*

The NGC Task Coordinator Module corresponds directly to the set of RCS level 4 (Task) Executor submodules plus the RCS level 4 World Model.

*The NGC Control Subsystem*

The NGC Machine Executive Module corresponds directly to the set of RCS level 3 (E-Move) Executor submodules plus the RCS level 3 World Model.

The NGC Profile Generator Module corresponds directly to the set of RCS level 2 (Primitive) task decomposition modules plus the RCS level 2 World Model.

The NGC Control Law module corresponds directly to the set of RCS level 1 (Servo) task decomposition modules plus the RCS level 1 World Model.

*The NGC Sensor/Effector Subsystem*

The NGC Sensor/Effector subsystem is the only feature of NGC not explicitly represented in RCS. RCS implicitly represents the Sensor/Effector subsystem by the interface between the RCS control system and its sensors and actuators.

On the other hand, RCS explicitly represents the processing of sensory information by a hierarchy of Sensory Processing modules that keep the RCS World Model knowledge database (NGC Information Base) updated with current sensory data. NGC does not explicitly represent Sensory Processing modules. They are implicitly represented in NGC by sensor subsystems.

For most current machine tool applications this is quite adequate. However, for some sophisticated robotic or inspection applications it may present problems. This is because many intelligent machine system applications require tightly coupled interactions between world model knowledge and sensory processing algorithms. These types of applications may involve "model-based processing" of sensory information, recursive estimation of world model parameters, or task driven control of sensory systems. These kinds of operations can not easily

be handled without explicitly representing Sensory Processing modules at each level of the control system hierarchy.

### 3.2.2 NGC Module Functionality

Fundamental to functionality is the representation and use of task knowledge. A task is a piece of work to be done, or an activity to be performed. For any TD module, there exists a set of tasks that the TD module knows how to do. Each task in this set can be assigned a name. The task vocabulary is the set of task names assigned to the set of tasks each TD module is capable of performing.

Knowledge of how to perform a task may be represented in a frame data structure. Task frames are data structures that capture a specification of the tasks that a given module can perform. This specification can be compared to a recipe in a cook book. The following is a template for a task frame:

| | |
|---|---|
| TASKNAME | Name of the task |
| GOAL | Event or condition that successfully terminates the task |
| OBJECT | Identification of thing to be acted upon |
| PARAMETERS | Priority<br>Status (e.g. active, waiting, inactive)<br>Timing (e.g. speed, completion time)<br>Coordinate system in which task is expressed<br>Stiffness matrices<br>Tolerances |
| AGENTS | Identification of subsystems that will perform the task |
| REQUIREMENTS | Feedback information required from the world model during the task<br>Tools, time, resources, and materials needed to perform the task<br>Enabling conditions that must be satisfied to begin or continue the task<br>Disabling conditions that will interrupt or abort the task activity |
| PROCEDURES | Pre-computed plans or scripts for executing the task<br>Planning algorithms<br>Functions that may be called<br>Emergency procedures for each disabling condition |

The TASKNAME is a string that identifies the type of activity to be performed.

The GOAL may be a vector that defines an attractor value, set point, or desired state to be achieved by the task. The goal may also be a map, graph, or geometric data structure that defines a desired "to-be" condition of an object, or arrangement of components.

The OBJECT attribute is a string that identifies what object is to be effected by the task. The value of the object attribute is a pointer to an entity frame in the world model that describes the object to be acted upon.

The values of PARAMETERS attributes uniquely identify a specific instantiation of the task, and define priority, speed, stiffness, coordinate frames, tolerances, etc. for the specific task.

AGENTS are the subsystems to be enlisted in the task activity.

The REQUIREMENTS section includes information required during the task. This may consist of a list of state variables, maps, and/or geometrical data structures that convey actual, or "as-is" conditions that currently exist in the world. Requirements may also include resources, tools, materials, time, and conditions needed for performing the task.

The PROCEDURES section contains either a set of pre-computed plans or scripts for decomposing the task, or one or more planning algorithms for generating a plan, or both. For example, the procedure section may contain a set of IF/THEN rules that select a plan appropriate to the "as-is" conditions reported by the world model. Alternatively, the procedure section may contain a planning algorithm that computes the difference between "to-be" and "as-is" conditions. This difference may then be treated as an error that the task planner attempts to reduce, or null through "Means/Ends Analysis" or A* search. Each subsystem planner would then develop a sequence of subtasks designed to minimize its subsystem error over an interval from the present to its planning horizon. In either case, each executor would act as a feedback controller, attempting to servo its respective subsystem to follow its plan. The procedure section also contains emergency procedures that can be executed immediately upon the detection of a disabling condition.

In plans involving concurrent job activity by different subsystems, there may be mutual constraints. For example, a start-event for a subtask activity in one subsystem may depend on the goal-event for a subtask activity in another subsystem. Some tasks may require concurrent and cooperative action by several subsystems. This requires that both planning and execution of subsystem plans be coordinated.[1]

In several RCS designs, human operators are an integral part of some computational nodes. This allows tasks that cannot yet be done automatically to be done in-

[1] The reader should not infer from this discussion or others throughout this report that all these difficult technical problems have been solved. The RCS architecture provides a framework wherein each of these problems can be isolated and explicitly represented and input/output relationships can be defined. The RCS architecture dies not specify the algorithms by which planning and constraint resolution are performed.

teractively by humans. In the NIST Automated Deburring and Chamfering System workstation, for example, the tasks of the fixturing subsystem are currently performed by a human operator. In future versions, these tasks will be performed automatically by a new node in the control system architecture.

The library of task frames that reside in each TD module define the capability of the TD module. The names of the task frames in the library define the set of task commands that TD module will accept. There, of course, may be several alternative ways that a task can be accomplished. Alternative task or job decompositions can be represented by an AND/OR graph in the procedure section of the task frame.

The agents, requirements, and procedures in the task frame specify for the TD module "how to do" commanded tasks. This information is a-priori resident in the task frame library of the TD module. The goal, object, and parameters specify "what to do", "on what object", "when", "how fast", etc.

The relationship between task frames and NML commands is shown in Figure 1. The NML command is expressed in the form of a command frame. The command frame consists of a task name, a goal to be achieved, an object to be acted on, and a set of parameters such as hand shake flags, how fast the task is to be performed, what priority it has, etc. The command frame specifies what to do. The task frame specifies how to do it. The command frame is communicated from a supervisor module to a subordinate. The task frames reside in the subordinate receiving the command. The name of the command must refer to a task frame in the library of the subordinate module, or the command cannot be executed.

When an NGC module inputs a task command, it searches its library of task frames to find a task name that matches the command name. Once a match is found, the goal, object, and parameter attributes from the command are transferred into the task frame. This activates the task frame, and as soon as the requirements listed in the task frame are met, the TD module can begin executing the task plan that carries out the job of task decomposition.

Task knowledge is typically difficult to discover, but once known, can be readily used and duplicated. For example, the proper way to mill a pocket, drill a hole, or fixture a part may be difficult to derive from first principles. However, once such knowledge is known and represented in a task frame, it is relatively easy to transform into executable code.

Because the task frame encapsulates both the functionality of the NGC modules and the information that must be communicated into and out of the NGC modules, a major part of the NIST effort in defining a strawman NML for the NGC has focused on the development of a methodology for creating task frames.

Once the library of task frames has been defined, a vocabulary of NML statements may be defined for each of the communication pathways between NGC modules.

A strawman subset of NML statements that apply to machine tool NGC modules is contained in Appendix C. A similar subset of NML was defined for robots in last year's final technical report to the AF NGC program office entitled A Strawman Neutral Manufacturing Language (NML) dated February 28, 1991. Another similar subset of NML can be defined for coordinate measuring machines, and programmable controllers.

A set of NGC commands for 3-axis machining has also been defined and are provided as Appendix D.

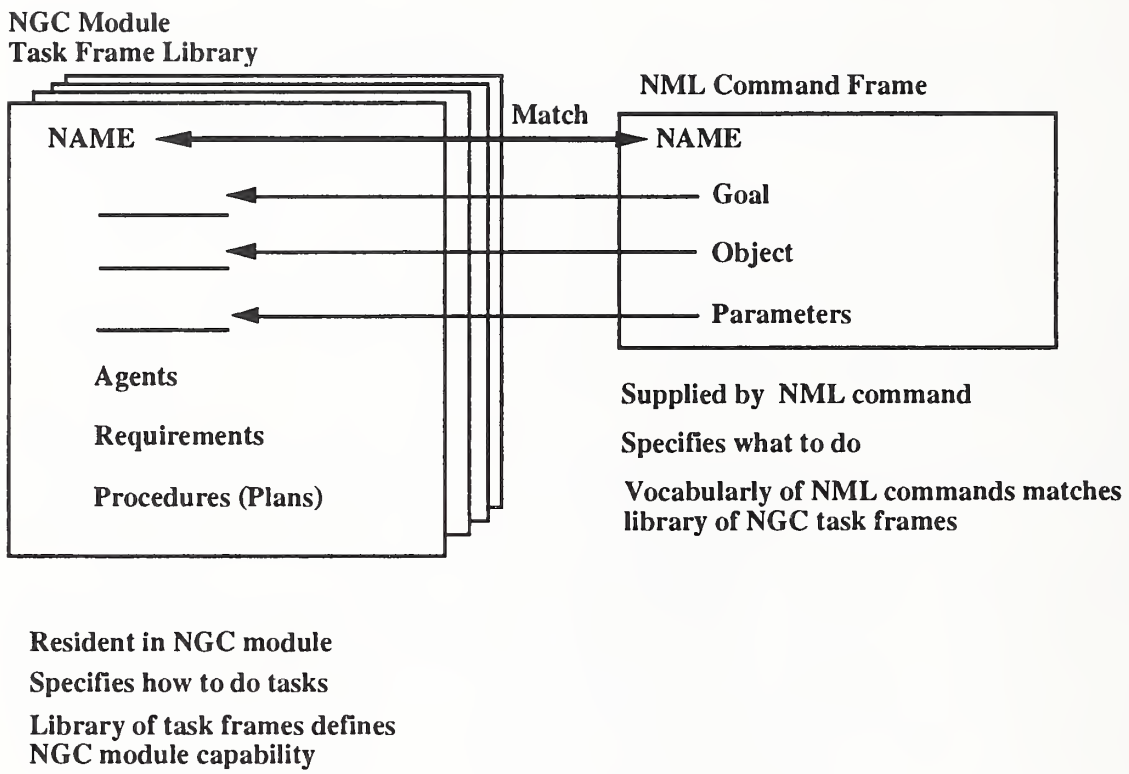



**Figure 1.** The relationship between commands and task frames

### 3.2.3 The Advanced Deburring and Chamfering System

NIST has developed automated finishing systems which use CAD data to generate robot programs, and has refined techniques to integrate sensory feedback into the real-time control of deburring and chamfering. Efforts are currently underway in the Advanced Deburring and Chamfering System (ADACS) to automate the finishing of aircraft engine components, such as turbine blades and rotors machined from inconel and titanium, whose hardness and complexity challenges commercial robotic finishing systems. The approach is to use a conventional T3-646 electric 6-axis robot as a coarse positioning device, while relying on an instrumented chamfering tool which is independently controlled to provide variable force and stiffness at the edges. NIST researchers are working jointly with United Technologies Research Center (UTRC) in this program, blending NIST's experience in robot control and sensor integration with the expertise of UTRC in the fabrication and finishing of aircraft engine components.

The ADACS workcell of the AMRF was selected by NIST as a testbed for studying the implementation of the NGC architecture in a robotic workcell. This workcell consists of the components shown in Figure 2. A complete set of task frames and command frames was designed for the ADACS scenario, and is included as Appendix E.

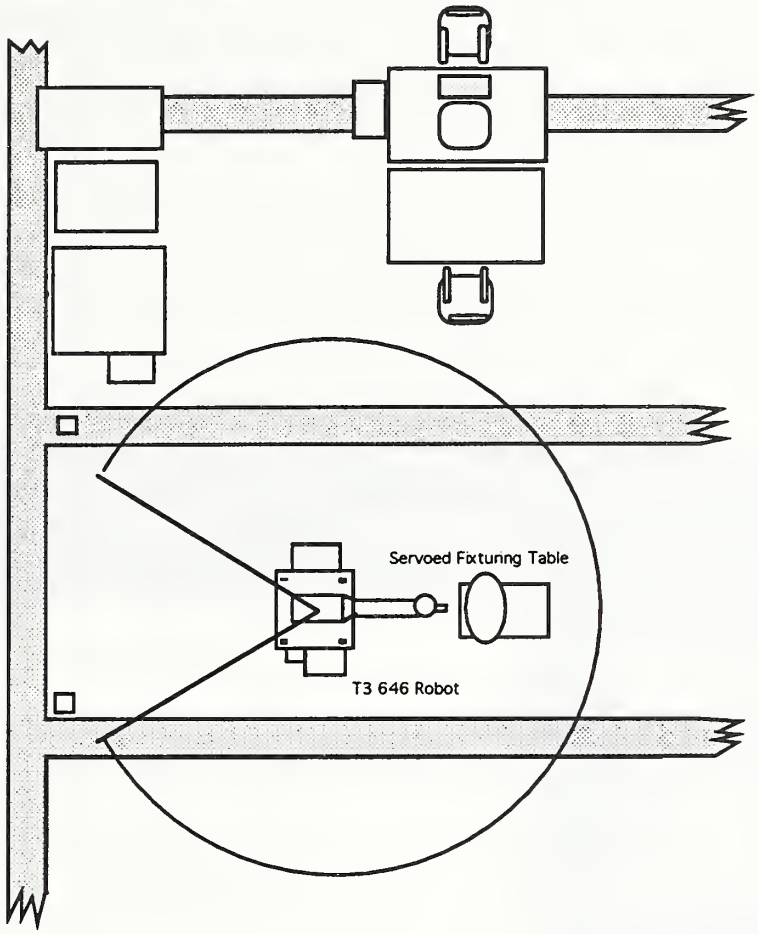



**Figure 2.** The Advanced Deburring and Chamfering System

Fred Proctor has been leading the effort to design and implement an open-architecture controller for the ADACS. His team has focused on the task knowledge representation, and the basic planning and execution engines in C and C++ which allow plans to be generated and executed in a consistent way across each level in the hierarchy. Additionally, a mathematics library for kinematics was developed and applied to robot trajectory planning for chamfering. Supporting our work in planning and execution, we have begun an Application Programming Interface (API) for communication, building on work performed at NIST, Martin Marietta, and elsewhere.

*ADACS Simulation and Process Planning*

A simulation of the ADACS has been developed to aid the off-line programming of chamfering trajectories. The simulation runs on a Silicon Graphics IRIS computer, in the CimStation modeling environment from Silma. The simulation includes the kinematics of the Cincinnati Milacron T3 robot and TriKinetics Adaptive

Deburring Tool (ADT), fixturing devices, and other objects in the ADACS. The robot and deburring tool are interfaced to a real-time control system which duplicates that in the real ADACS, allowing true robot and tool plans to be previewed in simulation before being applied on the shop floor. This off-line programming capability allows candidate plans to be quickly generated and tested without compromising safety or up-time. The display for the simulation has been extended so that commands received and the status of various workcell components can be easily viewed.

The ADACS simulation has been employed to develop Task Frames and their application to the Neutral Manufacturing Language. In particular, a set of Task Frames pursuant to the NIST Strawman Neutral Manufacturing Language have been enumerated for ADACS, and process plans for each have been interactively generated and tested on this simulator. A tool for viewing and editing ADACS task frames is shown in Figure 3.

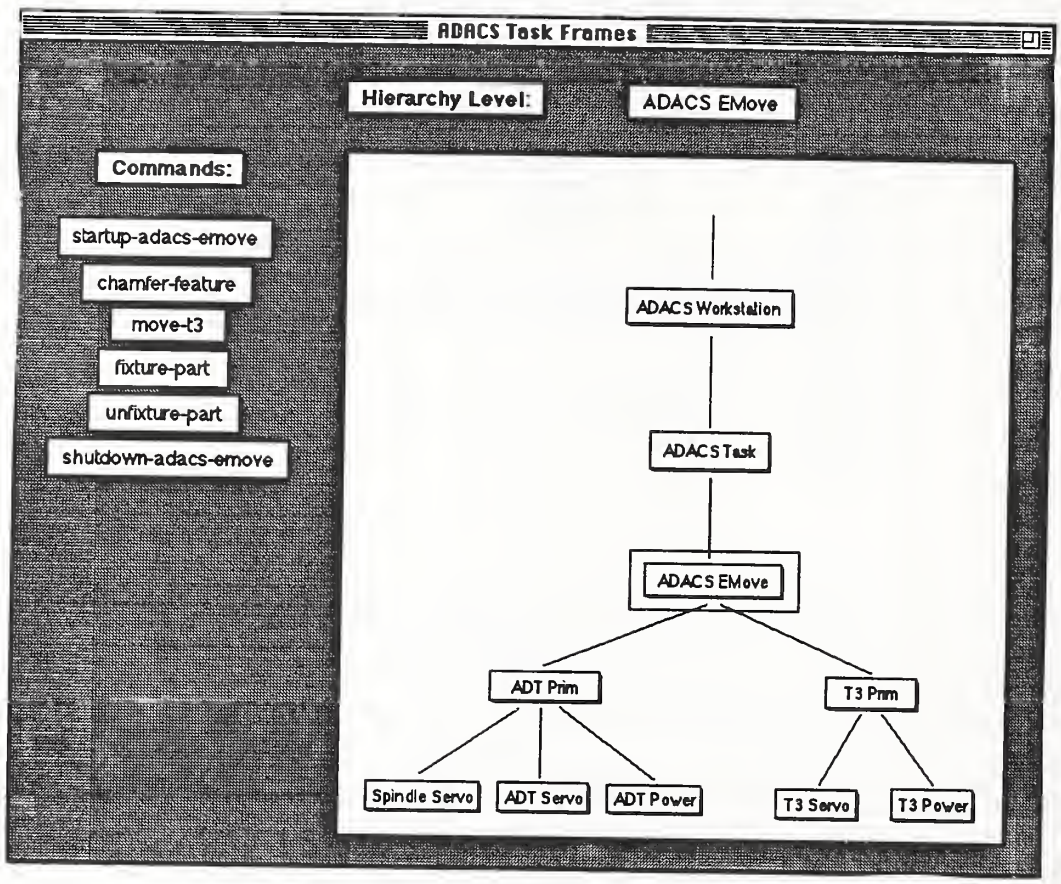



Figure 3. Macintosh HyperCard Tool for viewing and editing Task Frames

Nicholas Tarnoff has been developing a process planning system for ADACS. The objective of this work is to develop, assemble and integrate visually interactive tools for the development of process plans. The context for this work is an evolving methodology heavily based on consistent use of graphically interactive techniques.

The process planning system is able to simulate ADACS controller modules, and is interfaced to the workcell component simulator described above. The system also

provides various means of input and display for each of the simulated components. A command line interface is available for input to the simulated control modules. Control module status is displayed graphically. ADACS devices and parts are displayed as 3D solids and an ADACS force model is displayed in 2D. This system enables one to relatively quickly and iteratively explore task frames, NML, planning, world model, CASE, and other issues as part of actually developing a control system.

*Motion Planning for ADACS*

The robot Cartesian motion planner was improved to allow for the off-line generation of motion plans that can be quickly executed to reduce on-line planning delay. In normal operation, the plan for robot trajectories and tool force and speed is executed interpretively, and planning for complex sequences of moves slows down the processing. In the enhanced version of this planning and execution, complex plans may be generated off-line and converted to a larger but much "flatter" format, which can be executed with no planning. This option allows plans to be developed interactively, and optimized for execution when they have been demonstrated to run correctly.

*ADACS NGC Virtual Machine (VM)*

The purpose of the Virtual Machine is to logically abstract the application from the implementation. This abstraction is usually part of a broader concept, referred to in software terminology, as the Application Programmer Interface (API). Potential areas for API definitions include:

General System Software API Definitions

- cyclic process model
- communication using NML
- timing
- exception handling
- synchronization
- tasking

NGC API

- Data Modeling
- Entity Based : name-attribute, attribute-value
- Process Planning
- Feature Based Machining,
- Task Frames

We have designed and implemented a prototype API for NML communication, as well as an API for an RCS cyclic process model. We are exploring the timing needs of the API cyclic model.

Our implementation at NIST used an object-oriented approach to define any API. Within the definition of an API, a predefined set of methods are described that

the user instantiates when defining the API model. An instantiation of an API creates a process or communication object and then assigns the desired methods. For the cyclic process model, the method instantiation includes defining the following methods: initialization, step, run, suspend, resume, monitor (for debugging, diagnostics and timing configuration). The section below on NML describes its methods.

*NML and Virtual Machine Communication API*

As part of our effort to standardize the source code interface for an open-architecture controller, we have developed an Application Programmer Interface (API) for communication. Building on C source code given to us by Martin Marietta, we have extended the communications mechanism to include the NIST-developed Common Memory Manager (CMM). The CMM is a distributed client-server communications protocol which runs over a network, and has been demonstrated in the Automated Manufacturing Research Facility. We have ported the NIST CMM code to run on our VxWorks real-time operating system, in anticipation of including the process planning language ALPS. This way, a standard for plan representation and communication can be used by code developers working on this open-architecture controller.

For the communication API, the VM abstracts the communication process from the physical computational platform (PC, SUN, VAX, 680x0 target hardware, etc.) and the communication link (shared memory, Ethernet, etc.). We have implemented a NGC/VM communication model that abstracts the communication process from the user. All communication is done through READs and WRITEs to NML message buffers. An object-oriented approach is used to define the NML message buffers. Within the definition of an NML message buffer, a predefined set of methods are available (`init`, `read`, `write`, `getopt`, `setopt`, and `close`). The user configures an NML message descriptor, and then uses this descriptor for any future reads or writes.

These NML message buffers are configured according to a centralized configuration file format, (i.e., common memory reader-writer, TCP sockets, UDP sockets, etc.) that supplies the necessary underlying implementation parametrization (i.e., common memory may need an identification key or address, TCP sockets need host names and port numbers, etc.). This centralized configuration file could then be incorporated as a part of a System Integration and Configuration Tool (SICT). In evolutionary terms, a SICT should allow an NGC user to build a system around an available computer platform and communication links, without regard to the portability of the application code.

*Future NML Work*

In our labs we have implemented NML API that allow simple file-based reconfiguration (without system recompilation) of the system platform and links. We consider this an important step in evaluating NGC issues. Using this technology, we have developed a shell of a layered NGC architecture. We hope to flesh out the shell with appropriate technologies and other API in the future. Topics to be addressed in

the future include:

- understanding the role of timing and NML in configurations.
- developing plans/scripts to be executed by a process model.
- integrating a "seamless" graphical user interface (GUI) within the NML communication model.
- integrating other research activities into the NGC API model. We intend to use the NIST PDES/STEP existing process models and part planning methods as an information base part and process model API for an NGC.
- integrating task frame specification into the API model. Task frames offer a broader API view of the part planning.

## 3.3 Task 3: Define Strawman World Model Data Structures

Much of the power of the NGC/RCS/RCS architecture derives from knowledge stored in the world model. The world model knowledge database (KD) includes both a-priori information which is available to the intelligent system before action begins, and a-posterior knowledge which is gained from sensing the environment as action proceeds. The KD represents information about space, time, entities, events, states of the world, and laws of nature. Knowledge about space is represented in maps. Knowledge about entities, events, and states is represented in lists and frames. Knowledge about the laws of physics, chemistry, optics, and the rules of logic and mathematics is represented in the WM functions that generate predictions and simulate results of hypothetical actions. Laws of nature may be represented as formulae, or as IF/THEN rules of what happens under certain situations, such as when things are pushed, thrown, or dropped.

The world model also includes knowledge about the intelligent system itself, such as the values assigned to goal priorities, attribute values assigned to objects, and events; parameters defining kinematic and dynamic models of robot arms or machine tool stages; state variables describing internal pressure, temperature, clocks, fuel levels, body fluid chemistry; the state of all the currently executing processes in each of the computational modules; etc.

The correctness and consistency of world model knowledge is verified by sensors and sensory processing SP mechanisms that measure differences between world model predictions and sensory observations. These differences may be used by recursive estimation algorithms to keep the world model state variables the best estimates of the state of the world. Attention algorithms may be used to limit the number of state variables that must be kept up-to-date and any one time[2].

### 3.3.1 World Model Information Organization

Information in the world model knowledge database may be organized as state variables, system parameters, maps, and entity frames, as described below:

[2] The reader should not infer from this discussion or others throughout this report that all these difficult technical problems have been solved. The RCS architecture provides a framework wherein each of these problems can be isolated and explicitly represented and input/output relationships can be defined. The RCS architecture dies not specify the algorithms by which planning and constraint resolution are performed.

*State variables*

State variables define the current value of entity and system attributes. A state variable may define the state of a clock, the position, orientation, and velocity of a gripper, the position, velocity, and torque of an actuator, or the state of a computing module.

*System parameters*

System parameters define the kinematic and dynamic characteristics of the system being controlled, e.g., the inertia of objects, machines, and tools. System parameters may also define coordinate transforms necessary to transform commands and sensory information from one working coordinate system to another.

*Entity Frames*

An entity frame is a symbolic list structure, in which the ENTITY NAME is the list head, and in which knowledge about the entity is stored as attribute-value pairs (or attribute-list pairs). The world model contains a list of all the entities that the intelligent system knows about. A subset of this list is the set of current entities known to be present in any given situation. A subset of the list of current entities is the set of entities of attention. These are the entities that the system is currently acting upon, or is planning to act upon momentarily.

There are two types of entities: generic and specific. A generic entity is an example of a class of entities. A generic entity frame contains the attributes of its class. A specific entity is a particular instance of an entity. A specific entity frame inherits the attributes of its class. A template for an entity frame is shown below:

| | |
|---|---|
| ENTITY NAME | part id#, lot#, etc. |
| KIND | model# |
| TYPE | generic or specific |
| LEVEL | point, line, surface, object, group |
| POSITION | map location of center of mass (time, uncertainty) |
| ORIENTATION | coordinate axes directions (time, uncertainty) |
| COORDINATES | coordinate system of map |
| DYNAMICS | velocity,acceleration (time, uncertainty) |
| TRAJECTORY | sequence of positions (time, uncertainty) |
| GEOMETRY | center of gravity (uncertainty)<br>axis of symmetry (uncertainty)<br>size (uncertainty)<br>boundaries (uncertainty) |
| SUBENTITIES | pointers to lower level entities that make up named entity |

| | |
|---|---|
| PARENT ENTITY | pointer to higher level entity of which named entity is part |
| PROPERTIES | mass, color, hardness, smoothness, etc. |
| VALUES | sunk cost, value at completion |
| DUE DATE | date required |

Different levels of entities exist at different levels of the hierarchy. At level 1, entity frames describe points; at level 2, entity frames describe lines and vertices; at level 3, they describe surfaces; at level four, objects; and at level 5, groups; at level 6 and above, entity frames describe higher order groups. Figure 4 shows the object oriented structure of the entity database in the world model.

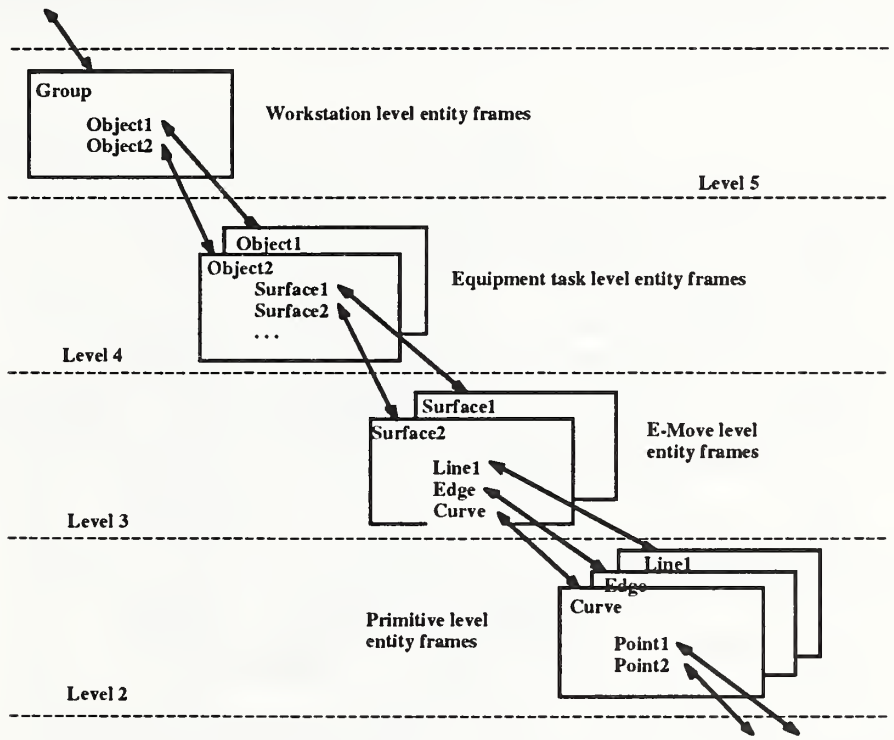



**Figure 4.** Object oriented relationships between entity frames at different hierarchical levels

### 3.3.2 Maps

Maps describe the distribution of entities in space. Each point, or pixel, on a map may have a pointer that points to the name of the entity that projects to that point on the map. A pixel may also have one or more attribute-value pairs. For example, a map pixel may have a brightness, or color (as in an image), or an altitude (as in a topographic map). A map may also be represented by a graph that indicates routes between locations, for example, the routes available to robot carts moving between workstations.

Any specific map is defined in a particular coordinate frame. There are three general types of map coordinate frames that are important: world coordinates, object coordinates, and egospheres. An egosphere is a spherical coordinate system with the

intelligent sensor system at the origin and properties of the world are projected onto the surface of the sphere. World, object, and egosphere coordinate frames are discussed more extensively in the paper "Outline for a Theory of Intelligence" in Appendix A.

### 3.3.3 Map-Entity Relationships

Map and entity representations may be cross referenced in the world model as shown in Figure 5. For example, each entity frame may contain a set of geometrical and state parameters that enables the world model to project that entity onto a map. The world model can thus compute the set of egosphere or world map pixels covered by an entity. By this means, entity parameters can be inherited by map pixels, and hence entity attributes can be overlaid on maps.

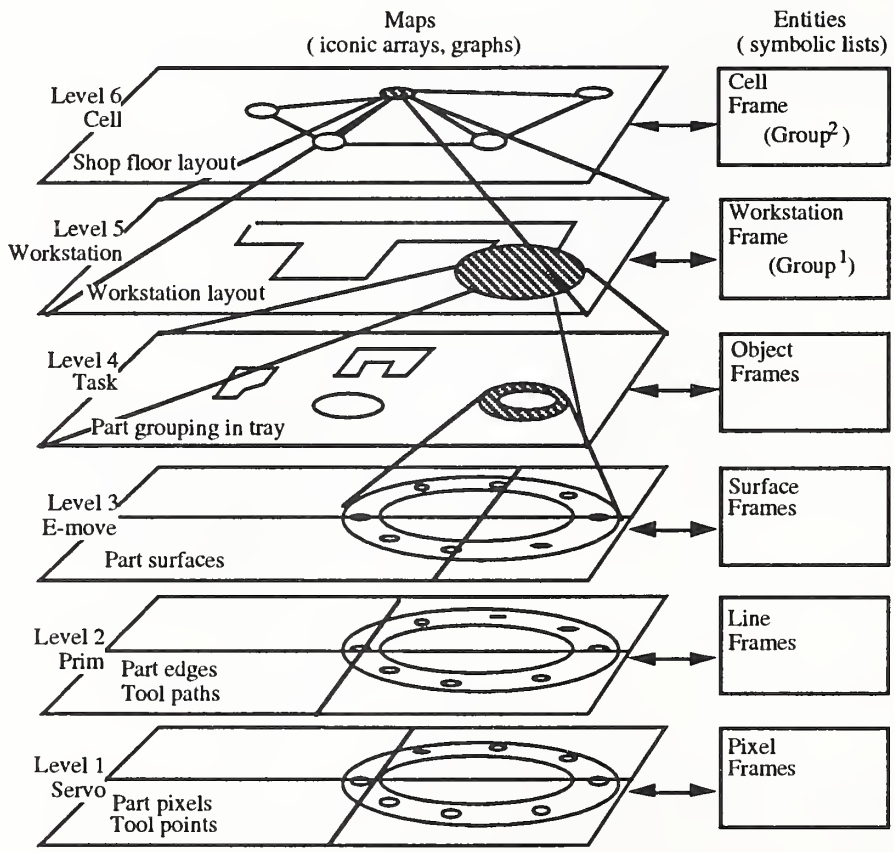



**Figure 5.** Map-entity relationships in a world model for manufacturing. The world model provides processes by which symbolic entity frames can be transformed into maps, and vice versa.

Conversely, each pixel on a map may have pointers to entities covered by that pixel. For example, a pixel may have a pointer to a point entity whose frame contains the projected distance or range to the point covered by that pixel. Each pixel may also have a pointer to a line entity frame indicating the position and orientation of an edge, line, or vertex covered by the pixel. Each pixel may also have a

pointer to a surface entity indicating position and orientation of a surface covered by the pixel. Each pixel may have a pointer to an object entity indicating the name of the object covered, and a pointer to a group entity indicating the name of the group covered.

The world model thus provides cross referencing between pixel maps and entity frames. Each level of world modeling can thus predict what objects will look like to sensors, and each level of sensory processing can compare sensory observations with world model predictions[3].

### 3.3.4 Example Knowledge Database for Manufacturing

An example of a knowledge database for a machine tool in a manufacturing workstation might be the following:

*Level 1*

| | |
|---|---|
| STATE VARIABLES | State clock and sync signals<br>State vector that defines the best estimate of the position, velocity, and force of each actuator (motor solenoid, servo valve, etc.), and each degree of freedom (slide, rotation)<br>Status of each switch or discrete actuator<br>State variables from the Human Interface |
| SYSTEM PARAMETERS | Joint limit margin for each actuator<br>Gravity compensation factors (if any)<br>Inertia matrix (if required)<br>Gain matrix for actuator controllers<br>Forward and inverse kinematic transform from tool-tip to actuator coordinates<br>Forward and inverse transform from sensor egosphere to end-effector egosphere (if required) |
| MAPS | Machine tool map overlaid with sensors and actuators showing sensor state variable indicated by bar-graphs, dials, colors, or numerical symbols |

*Level 2*

| | |
|---|---|
| STATE VARIABLES | State clock and sync signals<br>State vector defining the best estimate of tool position, velocity, force, etc.<br>Estimated time or distance to end of current trajectory segment<br>State variables from Human Interface |

[3] The reader should not infer from this discussion that such a cross-coupled systems have been fully implemented, or that it is even well understood how to implement such systems for real-time operations. What is described is the kind of cross-coupled system that will be necessary in order to achieve truly intelligent robotic systems. It seems likely that special purpose hardware and firmware will need to be developed. Clearly, much additional research remains to be done before these problems are solved. The RCS reference model architecture simply defines how such processes should be organized and what interfaces need to be defined.

| | |
|---|---|
| SYSTEM PARAMETERS | Forward and inverse force and velocity coordinate transform from part to tool-tip coordinates (including tool radius compensation)<br>Forward and inverse transform for sensory processing algorithms (if needed)<br>Machine tool dynamic model<br>Load dynamic model<br>Limits of travel in coordinate system of command<br>Positions of singularities<br>Position, velocity, and force limit |
| ENTITY FRAMES | Linear entity frames (lines, curves, trajectories, NURBS, vertices, etc.) for entities of attention (current planned tool-path, next tool-path, current observed tool-path, trace of following errors, etc.) |
| MAPS | Part feature map overlaid with projection of linear entities such as observed and planned tool paths, etc.<br>If there is a vision system, overlay of part feature map on image from camera |

*Level 3*

| | |
|---|---|
| STATE VARIABLES | State clock and sync signals<br>Best fit trajectories for observed poses, velocities, and forces of tool relative to part feature surfaces<br>Estimated time or distance to nearest part surface contact<br>Estimated clearance or distance to nearest obstacle surfaces<br>State variables from Human Interface |
| SYSTEM PARAMETERS | Forward and inverse transform from fixture to part coordinates<br>Limits of travel in coordinate system of command |
| ENTITY FRAMES | Surface entity frames defining position and geometry of "as-is" part features, "to-be" part features, and fixture surfaces, edges, alignment pins, etc. |
| MAPS | Part map overlaid with projection of as-is part feature surfaces, to-be part feature surfaces, tool-path swept-volumes, etc. showing graphic image of tool tip relative to surface of part being machine |

*Level 4*

| | |
|---|---|
| STATE VARIABLES | State clock and sync signals from other equipment<br>Best estimate observed degree of task completion<br>State of task enabling and disabling conditions<br>State variables from Human Interface |

| | |
|---|---|
| SYSTEM PARAMETERS | Task enabling and disabling conditions<br>Forward and inverse transform from machine tool coordinates to part fixture coordinates |
| ENTITY FRAMES | Object entity frames for objects of attention (robots, machine tools, as-is parts, to-be parts, part buffers, trays, fixtures, tool racks, free space, grippers, tools, fasteners, etc.) |
| MAPS | Machine tool workspace map overlaid with projections of objects of attention, plus swept volumes of planned sequences of part and tool motions |

*Level 5*

| | |
|---|---|
| STATE VARIABLES | State clock and sync signals from other equipment<br>Observed degree of task completion<br>State of task enabling and disabling conditions<br>State variables from Human Interface |
| SYSTEM PARAMETERS | Forward and inverse transforms from workstation to machine tool coordinates<br>Definition of task enabling and disabling conditions<br>Workstation task timing model<br>Cost/benefit evaluation function for planning results |
| ENTITY FRAMES | Group entity frames for workstation equipment arrangement, and groups of as-is and to-be groups of parts and tools arranged in trays, buffers, tool racks, etc. |
| MAPS | Workstation map overlaid with projections of equipment and groups of parts arranged on buffer tables and trays, tools in tool racks, etc. |

## 3.4 Task 4: Define Strawman NGC Robot Performance Measures and Measurements

The strawman performance measures and measurements work described here are designed for the ADACS project being used as a testbed for the support of this NGC effort. Working jointly with the United Technologies Research Center (UTRC), Nicholas Dagalakis has developed a set of performance measures for the robotic deburring and chamfering of hard metal parts, which is presently being tested. These performance measures can be divided into three categories:

### 3.4.1 Dynamic Response Performance

The objective of this performance measure is to evaluate the ability of the robot and the deburring tool holder to react to dynamic loads created by the deburring process. The amplitude of the displacement of the tool holder tip, in response to dynamic forces with amplitude equal to the maximum expected to be applied during

the deburring of a particular metal, should not exceed the maximum amplitude of the desired edge roughness.

During this test the frequency response of the robot and the deburring tool holder to external forces are determined using an instrumented hammer and accelerometers mounted at the the deburring tool tip. The resonance with the highest peak amplitude will determine the maximum amplitude of the desired edge roughness.

### 3.4.2 Static Stiffness

The sensors used for the measurement of the ADACS dynamic response performance are not capable of measuring the displacement of the tool holder tip, in response to dynamic forces at very low frequencies, typically below 1 Hz. For this reason a separate static stiffness test is necessary. During this test the tip of the tool is pushed through a load-cell and the resulting forces and displacement is monitored.

### 3.4.3 Continuous Path Off-line Programming Performance

The actual deburring operation involves the continuous movement of the tool along the edges of the part. The position and orientation of the edges is provided, in the ideal case, by off-line programming. The objective of this test is to evaluate the ability of the robot and the deburring tool holder to follow these edges as closely as possible. To perform this test a laser tracker interferometer is used to monitor the three dimensional space position of the tip of the tool. At the beginning of the test, the transformation between the laser tracker coordinate frame and the robot base frame is determined, then the robot arm is commanded to move the tip of the tool through various trajectories, characteristic of robot deburring, at orientations dictated by the assumed part geometry. The trajectory is followed ten times as specified by the relevant standards of the ISO and RIA/ANSI committees. The commanded and achieved trajectories are compared to determine performance figures of merit like accuracy, repeatability, cornering deviations, etc.

## 4 NIST-SUPPORTED NGC-RELATED ACTIVITIES

This section outlines activities supported with NIST internal funding and funds from other sponsors that are related to the NGC program. These activities are included here to show the degree of interest at NIST in the success of the NGC program and related efforts.

### 4.1 RCS Methodology

In order to address the need for developing NGC/RCS systems engineering guidelines, we have initiated an RCS Methodology project within the Robot Systems Division. Our long term objective is to produce an RCS Intelligent Systems Engineering Handbook (or text book) over the next several years. This Handbook will provide detailed procedures, design examples, and engineering tradeoff discussions, to guide systems engineers in designing NGC/RCS compliant control systems. The most difficult part of this process is arriving at a consensus among our own re-

searchers as to which of the engineering practices we currently use are generic to broad classes of applications projects and which are more application specific. We also are trying to identify which practices are driven by the technologies we currently use in implementation and which practices are technology and implementation independent and therefore less likely to change over time. Our second goal is to verify our guidelines in our own laboratory by applying the guidelines we develop to on-going projects within our division and by testing our hypotheses using test problems suitable for inclusion in the RCS Handbook.

To date John Michaloski has produced a preliminary draft of the RCS Methodology document. Work has begun on identifying the tenets of the RCS Methodology and in documenting the iterative procedures we use in designing RCS compliant control systems designs. Techniques for converting Task Frame specifications into executable code are also under investigation. During this process a number of issues have surfaced which need to be addressed through further research and experimentation. These issues have been captured in a data base with the intent of systematically addressing each issue during the course of our methodology project, and will be published in the "Handbook for Real-Time Intelligent System Design."

During this past summer, Dr. Alex Meystel of Drexel University was a guest scientist at NIST working with the Robot Systems Division. He was interested in studying the NIST RCS systems development approach and comparing it to his own hierarchically nested control systems approach, as well as other approaches found in the literature. He also expressed an interest in developing a course on how to apply the NIST RCS method. Dr. Meystel chaired a series of fact finding sessions at NIST which took the form of design review meetings for three of our on-going RCS applications projects: the ADACS project, the Army-sponsored Robotics Testbed (RT) project and the NASA-sponsored Flight Telerobot Servicer (FTS) NASREM project. These meetings were held on August 16 through 23 and August 26, 1991, respectively. Each of the project teams were asked to document their current designs using an initial Task Frame format and to review their design process and the rational for their particular design decisions. The purpose of these meetings was to make a first attempt at understanding which design procedures and decisions appear to be generic across RCS applications and which are application specific. Another purpose was to help refine a practical Task Frame format by studying the lessons learned in attempting to document three very different RCS applications using a single predefined Task Frame format.

## 4.2 CASE Tools

In a related effort, Rick Norcross developed a prototype documentation tool using Macintosh HyperCard software as the host environment. This tool allows the designer to document an NGC/RCS design by naming software modules and assigning them to specific levels within the RCS hierarchy using a Task Frame format. The inputs and outputs of each module can then be defined including task commands received from the next higher level and task commands sent to subordinate modules at the next lower level in the hierarchy. Test code to simulate the execution of the hierarchy can also be entered enabling the generation of execution timing diagrams that resemble a musical score. These kinds of tools will be needed in order

to specify the specific World Model data structures any particular application will need to support the task execution hierarchy.

Part of the RCS Methodology development effort has involved evaluating selected Computer Aided Software Engineering (CASE) tools. John Michaloski visited CIMFLEX Teknowledge on January 10 and 11, 1991 to discuss the possibility of using their tool (ABE and ABE-RT) for this purpose. Arrangements were made to install a copy of the tool at NIST in order to test its compatibility with NIST's RCS systems engineering approach.

The Robot Systems Division has been working this year with Advanced Research and Technology (ATR) of Laurel Maryland, on two other ongoing RCS automation projects: a DARPA submarine automation project, and a coal mining machine automation project sponsored by the Bureau of Mines. ATR has been investing IRAD funds to develop an RCS documentation tool to support the design and development of these two and other real-time control systems projects they have undertaken, using the RCS systems engineering approach. Their tool, which currently runs on a Macintosh computer, shows great promise. We are currently discussing a non-disclosure agreement with ATR which will allow us to use the tool at NIST as a beta test site.

## 4.3 PDES/STEP Activities

Dr. Tom Kramer has begun work on using PDES/STEP data representations in the NGC project. Geometric representations required will include part shapes (for workpieces, designs, and fixtures), material removal volumes, and chamfered-edges. Process plan representations will also be required. PDES/STEP boundary representations are expected to be adequate for part shapes. Experimental use of PDES/STEP B-splines for part and edge representations has begun.

In order to use a high-level, application-independent process planning language with NGC, it will be necessary to be able to merge such a language with low-level application-dependent suites of task and resource descriptions. EXPRESS, the official PDES/STEP information modeling language, does not make it easy to perform the needed mergers. A paper pointing out the problem and proposing changes to EXPRESS to solve the problem was written by Tom Kramer. The draft paper, Templates: A Needed but Missing Information Modeling Capability, is being circulated for comment.

Dr. Kramer has also written EXPRESS schemas for the ADACS workstation. One schema describes four levels of the ADACS control system (Task, E-Move, Primitive, and Servo). A second schema is for an ALPS representation of ADACS process plans. A third schema describes the geometry of the parts to be processed in the ADACS workstation. A fourth schema describes the topology of those parts. A copy of these EXPRESS schemas is included in Appendix F.

## 4.4 NCMS Next Generation Inspection Program

Jim Albus has been working with an NCMS Consortium to lay out plans for the development of an NGC compliant Next Generation Inspection System (an NGC controller for a coordinate measuring machine).

### 4.5 AMRF Automation Open House

A demonstration of the ADACS workstation featuring the Air Force NGC program involvement was prepared and presented at the NIST annual week-long AMRF Automation Open House. Demonstrations were conducted throughout the week of November 18th. Fred Proctor, Nicholas Tarnoff and Adam Jacoff contributed to this effort.

### 4.6 Intelligent Control System Architecture Workshops

Rick Quintero organized two control system architecture workshops this year. Working with the DOD Unmanned Ground Vehicles Project Office and the Army Human Engineering Laboratory the Second Workshop on Architectures for Real-Time Intelligent Control of Unmanned Vehicle Systems was held on January 10 and 11, 1991 in Aberdeen Maryland. Working with DOE the Second DOE/NIST Workshop on Common Architectures for Robotic Systems was held in Seattle, Washington on January 15 through 17, 1991. Both of these workshops explored many of the issues of how to define a reference model architecture for their respective problem domains which could be used as a basis for a standard systems engineering approach very similar to the NGC SOSAS approach.

## 5 PLANNED FUTURE ACTIVITIES

### 5.1 NGC Tasks 4-7 Demonstration Testbeds and Low End Controller

We are currently discussing with MMC and the Air Force plans to become an involved participant in the NGC Task 4 through 7 demonstration implementations and possibly the Low-End Controller effort. Negotiations between MMC, NIST and the Air Force will be taking place in October and November regarding the specific details of the NIST role.

NIST is proposing to work with Martin Marietta in support of their requirement for an off-site testbed for Tasks 4 through 7 of the NGC program. Our work for Tasks 4 and 5 will include defining the interfaces to a machine tool controller, allowing real-time sensor data from an external source to be used to modify cutter trajectories to increase accuracy. Work will be performed simultaneously at NIST and at Martin Marietta, in cooperation with two industry partners, demonstrating that the approach to improving machine tool accuracy is generic, and that the interfaces allow code to be ported between disparate platforms.

NIST's work for Tasks 6 and 7, the NGC Robot and Workstation Controllers, will likely take place in the Advanced Deburring and Chamfering System, and demonstrate the interfaces required to control both a robot and an actively compliant chamfering tool. This work will build upon the interfaces, subsystem, and module definitions resulting from our work with Tasks 4 and 5.

## 5.2 AMRF Enhanced Machine Tool Controller

We anticipate funding a portion of the work required for Tasks 4 and 5 as part of an AMRF program to develop an open-architecture machine tool controller. The goal, which coincides with that of NGC Task 4, is to work with an industry partner to develop a machine tool controller which presents well-documented and complete interfaces at each level of control, and to demonstrate that these interfaces allow for improved machine tool performance using real-time error compensation based on sensor feedback. The NIST funding for this project is $500K for FY92 plus about $500K worth of existing capital equipment.

## 5.3 Next Generation Inspection System (NGIS)

NIST intends to support the NGIS consortium with direct involvement in the overall system architecture design; software development for planning, control, and sensory processing; and sensor characterization. NIST will also construct a testbed consisting of a coordinate measuring machine and computer hardware for a NGC controller platform. NIST will contribute direct labor funding in the amount of $200K plus about $1,200K worth of existing capital equipment to this activity.

# Appendix A: NIST Papers on a Theoretical Model for Intelligent Systems

- Outline for a Theory of Intelligence

- A Reference Model Architecture for Intelligent Systems Design

# Outline for a Theory of Intelligence

James S. Albus

*Abstract*—**Intelligence is defined as that which produces successful behavior. Intelligence is assumed to result from natural selection. A model is proposed that integrates knowledge from research in both natural and artificial systems. The model consists of a hierarchical system architecture wherein: 1) control bandwidth decreases about an order of magnitude at each higher level, 2) perceptual resolution of spatial and temporal patterns contracts about an order-of-magnitude at each higher level, 3) goals expand in scope and planning horizons expand in space and time about an order-of-magnitude at each higher level, and 4) models of the world and memories of events expand their range in space and time by about an order-of-magnitude at each higher level. At each level, functional modules perform behavior generation (task decomposition planning and execution), world modeling, sensory processing, and value judgment. Sensory feedback control loops are closed at every level.**

## I. Introduction

MUCH IS UNKNOWN about intelligence, and much will remain beyond human comprehension for a very long time. The fundamental nature of intelligence is only dimly understood, and the elements of self consciousness, perception, reason, emotion, and intuition are cloaked in mystery that shrouds the human psyche and fades into the religious. Even the definition of intelligence remains a subject of controversy, and so must any theory that attempts to explain what intelligence is, how it originated, or what are the fundamental processes by which it functions.

Yet, much is known, both about the mechanisms and function of intelligence. The study of intelligent machines and the neurosciences are both extremely active fields. Many millions of dollars per year are now being spent in Europe, Japan, and the United States on computer integrated manufacturing, robotics, and intelligent machines for a wide variety of military and commercial applications. Around the world, researchers in the neurosciences are searching for the anatomical, physiological, and chemical basis of behavior.

Neuroanatomy has produced extensive maps of the interconnecting pathways making up the structure of the brain. Neurophysiology is demonstrating how neurons compute functions and communicate information. Neuropharmacology is discovering many of the transmitter substances that modify value judgments, compute reward and punishment, activate behavior, and produce learning. Psychophysics provides many clues as to how individuals perceive objects, events, time, and space, and how they reason about relationships between themselves and the external world. Behavioral psychology adds information about mental development, emotions, and behavior.

Research in learning automata, neural nets, and brain modeling has given insight into learning and the similarities and differences between neuronal and electronic computing processes. Computer science and artificial intelligence is probing the nature of language and image understanding, and has made significant progress in rule based reasoning, planning, and problem solving. Game theory and operations research have developed methods for decision making in the face of uncertainty. Robotics and autonomous vehicle research has produced advances in real-time sensory processing, world modeling, navigation, trajectory generation, and obstacle avoidance. Research in automated manufacturing and process control has produced intelligent hierarchical controls, distributed databases, representations of object geometry and material properties, data driven task sequencing, network communications, and multiprocessor operating systems. Modern control theory has developed precise understanding of stability, adaptability, and controllability under various conditions of feedback and noise. Research in sonar, radar, and optical signal processing has developed methods for fusing sensory input from multiple sources, and assessing the believability of noisy data.

Progress is rapid, and there exists an enormous and rapidly growing literature in each of the previous fields. What is lacking is a general theoretical model of intelligence that ties all these separate areas of knowledge into a unified framework. This paper is an attempt to formulate at least the broad outlines of such a model.

The ultimate goal is a general theory of intelligence that encompasses both biological and machine instantiations. The model presented here incorporates knowledge gained from many different sources and the discussion frequently shifts back and forth between natural and artificial systems. For example, the definition of intelligence in Section II addresses both natural and artificial systems. Section III treats the origin and function of intelligence from the standpoint of biological evolution. In Section IV, both natural and artificial system elements are discussed. The system architecture described in Sections V–VII derives almost entirely from research in robotics and control theory for devices ranging from undersea vehicles to automatic factories. Sections VIII–XI on behavior generation, Sections XII and XIII on world modeling, and Section XIV on sensory processing are elaborations of the system architecture of Section V–VII. These sections all contain numerous references to neurophysiological, psychological, and psychophysical phenomena that support the model, and frequent analogies are drawn between biological and artificial

Manuscript received March 16, 1990; revised November 16, 1990.

The author is with the Robot Systems Division Center for Manufacturing Engineering, National Institute of Standards and Technology, Gaithersburg, MD 20899.

IEEE Log Number 9042583.

systems. The value judgments, described in Section XV, are mostly based on the neurophysiology of the limbic system and the psychology of emotion. Section XVI on neural computation and Section XVII on learning derive mostly from neural net research.

The model is described in terms of definitions, axioms, theorems, hypotheses, conjectures, and arguments in support of them. Axioms are statements that are assumed to be true without proof. Theorems are statements that the author feels could be demonstrated true by existing logical methods or empirical evidence. Few of the theorems are proven, but each is followed by informal discussions that support the theorem and suggest arguments upon which a formal proof might be constructed. Hypotheses are statements that the author feels probably could be demonstrated through future research. Conjectures are statements that the author feels might be demonstrable.

## II. Definition Of Intelligence

In order to be useful in the quest for a general theory, the definition of intelligence must not be limited to behavior that is not understood. A useful definition of intelligence should span a wide range of capabilities, from those that are well understood, to those that are beyond comprehension. It should include both biological and machine embodiments, and these should span an intellectual range from that of an insect to that of an Einstein, from that of a thermostat to that of the most sophisticated computer system that could ever be built. The definition of intelligence should, for example, include the ability of a robot to spotweld an automobile body, the ability of a bee to navigate in a field of wild flowers, a squirrel to jump from limb to limb, a duck to land in a high wind, and a swallow to work a field of insects. It should include what enables a pair of blue jays to battle in the branches for a nesting site, a pride of lions to pull down a wildebeest, a flock of geese to migrate south in the winter. It should include what enables a human to bake a cake, play the violin, read a book, write a poem, fight a war, or invent a computer.

At a minimum, intelligence requires the ability to sense the environment, to make decisions, and to control action. Higher levels of intelligence may include the ability to recognize objects and events, to represent knowledge in a world model, and to reason about and plan for the future. In advanced forms, intelligence provides the capacity to perceive and understand, to choose wisely, and to act successfully under a large variety of circumstances so as to survive, prosper, and reproduce in a complex and often hostile environment.

From the viewpoint of control theory, intelligence might be defined as a knowledgeable "helmsman of behavior". Intelligence is the integration of knowledge and feedback into a sensory-interactive goal-directed control system that can make plans, and generate effective, purposeful action directed toward achieving them.

From the viewpoint of psychology, intelligence might be defined as a behavioral strategy that gives each individual a means for maximizing the likelihood of propagating its own genes. Intelligence is the integration of perception, reason, emotion, and behavior in a sensing, perceiving, knowing, caring, planning, acting system that can succeed in achieving its goals in the world.

For the purposes of this paper, intelligence will be defined as the ability of a system to act appropriately in an uncertain environment, where appropriate action is that which increases the probability of success, and success is the achievement of behavioral subgoals that support the system's ultimate goal.

Both the criteria of success and the systems ultimate goal are defined external to the intelligent system. For an intelligent machine system, the goals and success criteria are typically defined by designers, programmers, and operators. For intelligent biological creatures, the ultimate goal is gene propagation, and success criteria are defined by the processes of natural selection.

*Theorem:* There are degrees, or levels, of intelligence, and these are determined by: 1) the computational power of the system's brain (or computer), 2) the sophistication of algorithms the system uses for sensory processing, world modeling, behavior generating, value judgment, and global communication, and 3) the information and values the system has stored in its memory.

Intelligence can be observed to grow and evolve, both through growth in computational power, and through accumulation of knowledge of how to sense, decide, and act in a complex and changing world. In artificial systems, growth in computational power and accumulation of knowledge derives mostly from human hardware engineers and software programmers. In natural systems, intelligence grows, over the lifetime of an individual, through maturation and learning; and over intervals spanning generations, through evolution.

Note that learning is not required in order to be intelligent, only to become more intelligent as a result of experience. Learning is defined as consolidating short-term memory into long-term memory, and exhibiting altered behavior because of what was remembered. In Section X, learning is discussed as a mechanism for storing knowledge about the external world, and for acquiring skills and knowledge of how to act. It is, however, assumed that many creatures can exhibit intelligent behavior using instinct, without having learned anything.

## III. The Origin And Function Of Intelligence

*Theorem:* Natural intelligence, like the brain in which it appears, is a result of the process of natural selection.

The brain is first and foremost a control system. Its primary function is to produce successful goal-seeking behavior in finding food, avoiding danger, competing for territory, attracting sexual partners, and caring for offspring. All brains that ever existed, even those of the tiniest insects, generate and control behavior. Some brains produce only simple forms of behavior, while others produce very complex behaviors. Only the most recent and highly developed brains show any evidence of abstract thought.

*Theorem:* For each individual, intelligence provides a mechanism for generating biologically advantageous behavior.

Intelligence improves an individual's ability to act effectively and choose wisely between alternative behaviors. All

else being equal, a more intelligent individual has many advantages over less intelligent rivals in acquiring choice territory, gaining access to food, and attracting more desirable breeding partners. The intelligent use of aggression helps to improve an individual's position in the social dominance hierarchy. Intelligent predation improves success in capturing prey. Intelligent exploration improves success in hunting and establishing territory. Intelligent use of stealth gives a predator the advantage of surprise. Intelligent use of deception improves the prey's chances of escaping from danger.

Higher levels of intelligence produce capabilities in the individual for thinking ahead, planning before acting, and reasoning about the probable results of alternative actions. These abilities give to the more intelligent individual a competitive advantage over the less intelligent in the competition for survival and gene propagation. Intellectual capacities and behavioral skills that produce successful hunting and gathering of food, acquisition and defense of territory, avoidance and escape from danger, and bearing and raising offspring tend to be passed on to succeeding generations. Intellectual capabilities that produce less successful behaviors reduce the survival probability of the brains that generate them. Competition between individuals thus drives the evolution of intelligence within a species.

*Theorem:* For groups of individuals, intelligence provides a mechanism for cooperatively generating biologically advantageous behavior.

The intellectual capacity to simply congregate into flocks, herds, schools, and packs increases the number of sensors watching for danger. The ability to communicate danger signals improves the survival probability of all individuals in the group. Communication is most advantageous to those individuals who are the quickest and most discriminating in the recognition of danger messages, and most effective in responding with appropriate action. The intelligence to cooperate in mutually beneficial activities such as hunting and group defense increases the probability of gene propagation for all members of the group.

All else being equal, the most intelligent individuals and groups within a species will tend to occupy the best territory, be the most successful in social competition, and have the best chances for their offspring surviving. All else being equal, more intelligent individuals and groups will win out in serious competition with less intelligent individuals and groups.

Intelligence is, therefore, the product of continuous competitive struggles for survival and gene propagation that has taken place between billions of brains, over millions of years. The results of those struggles have been determined in large measure by the intelligence of the competitors.

### A. Communication and Language

*Definition:* Communication is the transmission of information between intelligent systems.

*Definition:* Language is the means by which information is encoded for purposes of communication.

Language has three basic components: vocabulary, syntax, and semantics. Vocabulary is the set of words in the language. Words may be represented by symbols. Syntax, or grammar, is the set of rules for generating strings of symbols that form sentences. Semantics is the encoding of information into meaningful patterns, or messages. Messages are sentences that convey useful information.

Communication requires that information be: 1) encoded, 2) transmitted, 3) received, 4) decoded, and 5) understood. Understanding implies that the information in the message has been correctly decoded and incorporated into the world model of the receiver.

Communication may be either intentional or unintentional. Intentional communication occurs as the result of a sender executing a task whose goal it is to alter the knowledge or behavior of the receiver to the benefit of the sender. Unintentional communication occurs when a message is unintentionally sent, or when an intended message is received and understood by someone other than the intended receiver. Preventing an enemy from receiving and understanding communication between friendly agents can often be crucial to survival.

Communication and language are by no means unique to human beings. Virtually all creatures, even insects, communicate in some way, and hence have some form of language. For example, many insects transmit messages announcing their identity and position. This may be done acoustically, by smell, or by some visually detectable display. The goal may be to attract a mate, or to facilitate recognition and/or location by other members of a group. Species of lower intelligence, such as insects, have very little information to communicate, and hence have languages with only a few of what might be called words, with little or no grammar. In many cases, language vocabularies include motions and gestures (i.e., body or sign language) as well as acoustic signals generated by variety of mechanisms from stamping the feet, to snorts, squeals, chirps, cries, and shouts.

*Theorem:* In any species, language evolves to support the complexity of messages that can be generated by the intelligence of that species.

Depending on its complexity, a language may be capable of communicating many messages, or only a few. More intelligent individuals have a larger vocabulary, and are quicker to understand and act on the meaning of messages.

*Theorem:* To the receiver, the benefit, or value, of communication is roughly proportional to the product of the amount of information contained in the message, multiplied by the ability of the receiver to understand and act on that information, multiplied by the importance of the act to survival and gene propagation of the receiver. To the sender, the benefit is the value of the receiver's action to the sender, minus the danger incurred by transmitting a message that may be intercepted by, and give advantage to, an enemy.

Greater intelligence enhances both the individual's and the group's ability to analyze the environment, to encode and transmit information about it, to detect messages, to recognize their significance, and act effectively on information received. Greater intelligence produces more complex languages capable of expressing more information, i.e., more messages with more shades of meaning.

In social species, communication also provides the basis

for societal organization. Communication of threats that warn of aggression can help to establish the social dominance hierarchy, and reduce the incidence of physical harm from fights over food, territory, and sexual partners. Communication of alarm signals indicate the presence of danger, and in some cases, identify its type and location. Communication of pleas for help enables group members to solicit assistance from one another. Communication between members of a hunting pack enable them to remain in formation while spread far apart, and hence to hunt more effectively by cooperating as a team in the tracking and killing of prey.

Among humans, primitive forms of communication include facial expressions, cries, gestures, body language, and pantomime. The human brain is, however, capable of generating ideas of much greater complexity and subtlety than can be expressed through cries and gestures. In order to transmit messages commensurate with the complexity of human thought, human languages have evolved grammatical and semantic rules capable of stringing words from vocabularies consisting of thousands of entries into sentences that express ideas and concepts with exquisitely subtle nuances of meaning. To support this process, the human vocal apparatus has evolved complex mechanisms for making a large variety of sounds.

### B. Human Intelligence and Technology

Superior intelligence alone made man a successful hunter. The intellectual capacity to make and use tools, weapons, and spoken language made him the most successful of all predators. In recent millennia, human levels of intelligence have led to the use of fire, the domestication of animals, the development of agriculture, the rise of civilization, the invention of writing, the building of cities, the practice of war, the emergence of science, and the growth of industry. These capabilities have extremely high gene propagation value for the individuals and societies that possess them relative to those who do not. Intelligence has thus made modern civilized humans the dominant species on the planet Earth.

For an individual human, superior intelligence is an asset in competing for position in the social dominance hierarchy. It conveys advantage for attracting and winning a desirable mate, in raising a large, healthy, and prosperous family, and seeing to it that one's offspring are well provided for. In competition between human groups, more intelligent customs and traditions, and more highly developed institutions and technology, lead to the dominance of culture and growth of military and political power. Less intelligent customs, traditions, and practices, and less developed institutions and technology, lead to economic and political decline and eventually to the demise of tribes, nations, and civilizations.

## IV. The Elements Of Intelligence

*Theorem:* There are four system elements of intelligence: sensory processing, world modeling, behavior generation, and value judgment. Input to, and output from, intelligent systems are via sensors and actuators.

*1) Actuators:* Output from an intelligent system is produced by actuators that move, exert forces, and position arms, legs, hands, and eyes. Actuators generate forces to point sensors, excite transducers, move manipulators, handle tools, steer and propel locomotion. An intelligent system may have tens, hundreds, thousands, even millions of actuators, all of which must be coordinated in order to perform tasks and accomplish goals. Natural actuators are muscles and glands. Machine actuators are motors, pistons, valves, solenoids, and transducers.

*2) Sensors:* Input to an intelligent system is produced by sensors, which may include visual brightness and color sensors; tactile, force, torque, position detectors; velocity, vibration, acoustic, range, smell, taste, pressure, and temperature measuring devices. Sensors may be used to monitor both the state of the external world and the internal state of the intelligent system itself. Sensors provide input to a sensory processing system.

*3) Sensory Processing:* Perception takes place in a sensory processing system element that compares sensory observations with expectations generated by an internal world model. Sensory processing algorithms integrate similarities and differences between observations and expectations over time and space so as to detect events and recognize features, objects, and relationships in the world. Sensory input data from a wide variety of sensors over extended periods of time are fused into a consistent unified perception of the state of the world. Sensory processing algorithms compute distance, shape, orientation, surface characteristics, physical and dynamical attributes of objects and regions of space. Sensory processing may include recognition of speech and interpretation of language and music.

*4) World Model:* The world model is the intelligent system's best estimate of the state of the world. The world model includes a database of knowledge about the world, plus a database management system that stores and retrieves information. The world model also contains a simulation capability that generates expectations and predictions. The world model thus can provide answers to requests for information about the present, past, and probable future states of the world. The world model provides this information service to the behavior generation system element, so that it can make intelligent plans and behavioral choices, to the sensory processing system element, in order for it to perform correlation, model matching, and model based recognition of states, objects, and events, and to the value judgment system element in order for it to compute values such as cost, benefit, risk, uncertainty, importance, attractiveness, etc. The world model is kept up-to-date by the sensory processing system element.

*5) Value Judgment:* The value judgment system element determines what is good and bad, rewarding and punishing, important and trivial, certain and improbable. The value judgment system evaluates both the observed state of the world and the predicted results of hypothesized plans. It computes costs, risks, and benefits both of observed situations and of planned activities. It computes the probability of correctness and assigns believability and uncertainty parameters to state variables. It also assigns attractiveness, or repulsiveness to objects, events, regions of space, and other creatures. The value judgment system thus provides the basis for making

decisions—for choosing one action as opposed to another, or for pursuing one object and fleeing from another. Without value judgments, any biological creature would soon be eaten by others, and any artificially intelligent system would soon be disabled by its own inappropriate actions.

*6) Behavior Generation:* Behavior results from a behavior generating system element that selects goals, and plans and executes tasks. Tasks are recursively decomposed into subtasks, and subtasks are sequenced so as to achieve goals. Goals are selected and plans generated by a looping interaction between behavior generation, world modeling, and value judgment elements. The behavior generating system hypothesizes plans, the world model predicts the results of those plans, and the value judgment element evaluates those results. The behavior generating system then selects the plans with the highest evaluations for execution. The behavior generating system element also monitors the execution of plans, and modifies existing plans whenever the situation requires.

Each of the system elements of intelligence are reasonably well understood. The phenomena of intelligence, however, requires more than a set of disconnected elements. Intelligence requires an interconnecting system architecture that enables the various system elements to interact and communicate with each other in intimate and sophisticated ways.

A system architecture is what partitions the system elements of intelligence into computational modules, and interconnects the modules in networks and hierarchies. It is what enables the behavior generation elements to direct sensors, and to focus sensory processing algorithms on objects and events worthy of attention, ignoring things that are not important to current goals and task priorities. It is what enables the world model to answer queries from behavior generating modules, and make predictions and receive updates from sensory processing modules. It is what communicates the value state-variables that describe the success of behavior and the desirability of states of the world from the value judgment element to the goal selection subsystem.

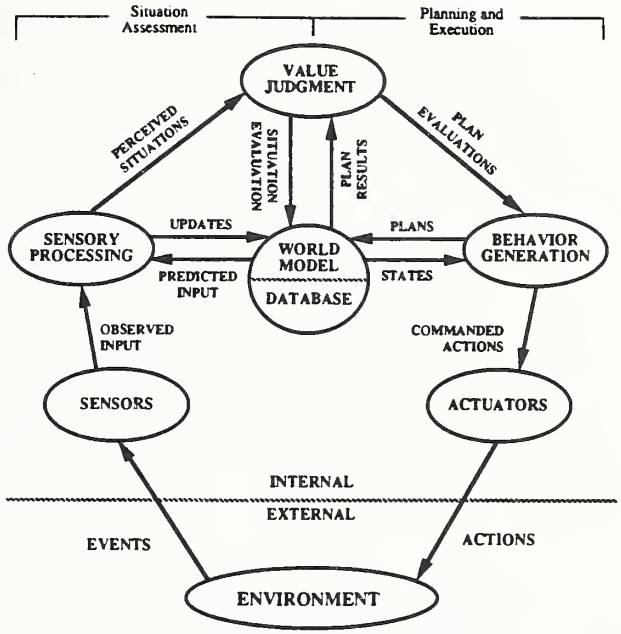



Fig. 1. Elements of intelligence and the functional relationships between them.

## V. A Proposed Architecture For Intelligent Systems

A number of system architectures for intelligent machine systems have been conceived, and a few implemented. [1]–[15] The architecture for intelligent systems that will be proposed here is largely based on the real-time control system (RCS) that has been implemented in a number of versions over the past 13 years at the National Institute for Standards and Technology (NIST, formerly NBS). RCS was first implemented by Barbera for laboratory robotics in the mid 1970's [7] and adapted by Albus, Barbera, and others for manufacturing control in the NIST Automated Manufacturing Research Facility (AMRF) during the early 1980's [11], [12]. Since 1986, RCS has been implemented for a number of additional applications, including the NBS/DARPA Multiple Autonomous Undersea Vehicle (MAUV) project [13], the Army Field Material Handling Robot, and the Army TMAP and TEAM semiautonomous land vehicle projects. RCS also forms the basis of the NASA/NBS Standard Reference Model Telerobot Control System Architecture (NASREM) being used on the space station Flight Telerobotic Servicer [14] and the Air Force Next Generation Controller.

The proposed system architecture organizes the elements of intelligence so as to create the functional relationships and information flow shown in Fig. 1. In all intelligent systems, a sensory processing system processes sensory information to acquire and maintain an internal model of the external world. In all systems, a behavior generating system controls actuators so as to pursue behavioral goals in the context of the perceived world model. In systems of higher intelligence, the behavior generating system element may interact with the world model and value judgment system to reason about space and time, geometry and dynamics, and to formulate or select plans based on values such as cost, risk, utility, and goal priorities. The sensory processing system element may interact with the world model and value judgment system to assign values to perceived entities, events, and situations.

The proposed system architecture replicates and distributes the relationships shown in Fig. 1 over a hierarchical computing structure with the logical and temporal properties illustrated in Fig. 2. On the left is an organizational hierarchy wherein computational nodes are arranged in layers like command posts in a military organization. Each node in the organizational hierarchy contains four types of computing modules: behavior generating (BG), world modeling (WM), sensory processing (SP), and value judgment (VJ) modules. Each chain of command in the organizational hierarchy, from each actuator and each sensor to the highest level of control, can be represented by a computational hierarchy, such as is shown in the center of Fig. 2.

At each level, the nodes, and computing modules within the nodes, are richly interconnected to each other by a communications system. Within each computational node, the communication system provides intermodule communications

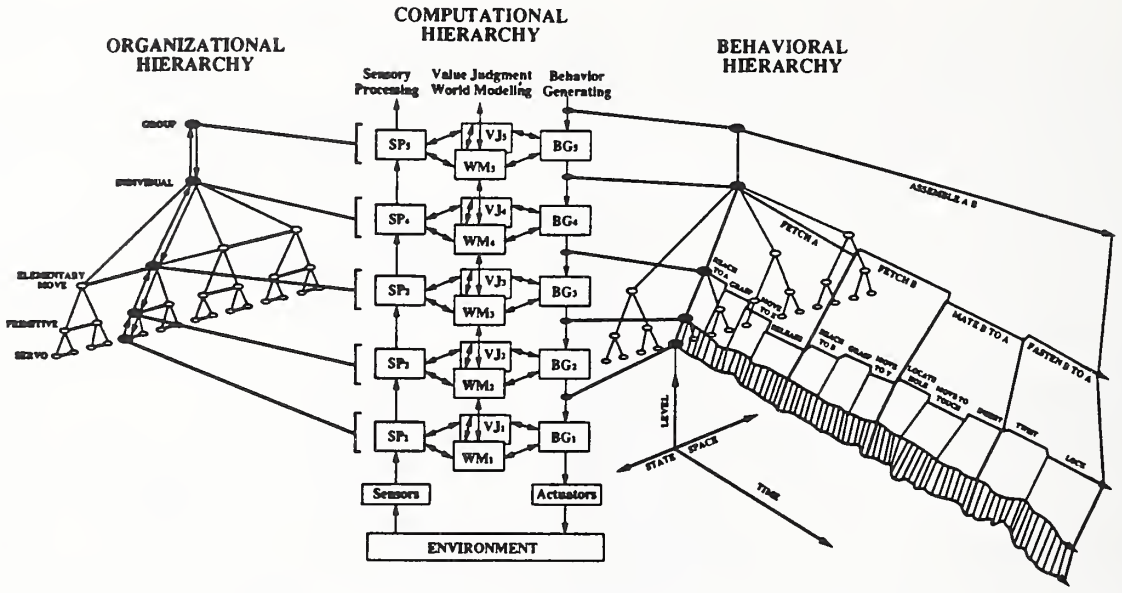



Fig. 2. Relationships in hierarchical control systems, On the left is an organizational hierarchy consisting of a tree of command centers, each of which possesses one supervisor and one or more subordinates. In the center is a computational hierarchy consisting of BG, WM, SP, and VJ modules. Each actuator and each sensors is serviced by a computational hierarchy. On the right is a behavioral hierarchy consisting of trajectories through state–time–space. Commands at a each level can be represented by vectors, or points in state–space. Sequences of commands and be represented as trajectories through state–time–space.

of the type shown in Fig. 1. Queries and task status are communicated from BG modules to WM modules. Retrievals of information are communicated from WM modules back to the BG modules making the queries. Predicted sensory data is communicated from WM modules to SP modules. Updates to the world model are communicated from SP to WM modules. Observed entities, events, and situations are communicated from SP to VJ modules. Values assigned to the world model representations of these entities, events, and situations are communicated from VJ to WM modules. Hypothesized plans are communicated from BG to WM modules. Results are communicated from WM to VJ modules. Evaluations are communicated from VJ modules back to the BG modules that hypothesized the plans.

The communications system also communicates between nodes at different levels. Commands are communicated downward from supervisor BG modules in one level to subordinate BG modules in the level below. Status reports are communicated back upward through the world model from lower level subordinate BG modules to the upper level supervisor BG modules from which commands were received. Observed entities, events, and situations detected by SP modules at one level are communicated upward to SP modules at a higher level. Predicted attributes of entities, events, and situations stored in the WM modules at a higher level are communicated downward to lower level WM modules. Output from the bottom level BG modules is communicated to actuator drive mechanisms. Input to the bottom level SP modules is communicated from sensors.

The communications system can be implemented in a variety of ways. In a biological brain, communication is mostly via neuronal axon pathways, although some messages are communicated by hormones carried in the bloodstream. In artificial systems, the physical implementation of communications functions may be a computer bus, a local area network, a common memory, a message passing system, or some combination thereof. In either biological or artificial systems, the communications system may include the functionality of a communications processor, a file server, a database management system, a question answering system, or an indirect addressing or list processing engine. In the system architecture proposed here, the input/output relationships of the communications system produce the effect of a virtual global memory, or blackboard system [15].

The input command string to each of the BG modules at each level generates a trajectory through state-space as a function of time. The set of all command strings create a behavioral hierarchy, as shown on the right of Fig. 2. Actuator output trajectories (not shown in Fig. 2) correspond to observable output behavior. All the other trajectories in the behavioral hierarchy constitute the deep structure of behavior [16].

## VI. Hierarchical Versus Horizontal

Fig. 3 shows the organizational hierarchy in more detail, and illustrates both the hierarchical and horizontal relationships involved in the proposed architecture. The architecture is hierarchical in that commands and status feedback flow hierarchically up and down a behavior generating chain of command. The architecture is also hierarchical in that sensory processing and world modeling functions have hierarchical levels of temporal and spatial aggregation.

The architecture is horizontal in that data is shared horizontally between heterogeneous modules at the same level. At each hierarchical level, the architecture is horizontally interconnected by wide-bandwidth communication pathways between BG, WM, SP, and VJ modules in the same node,

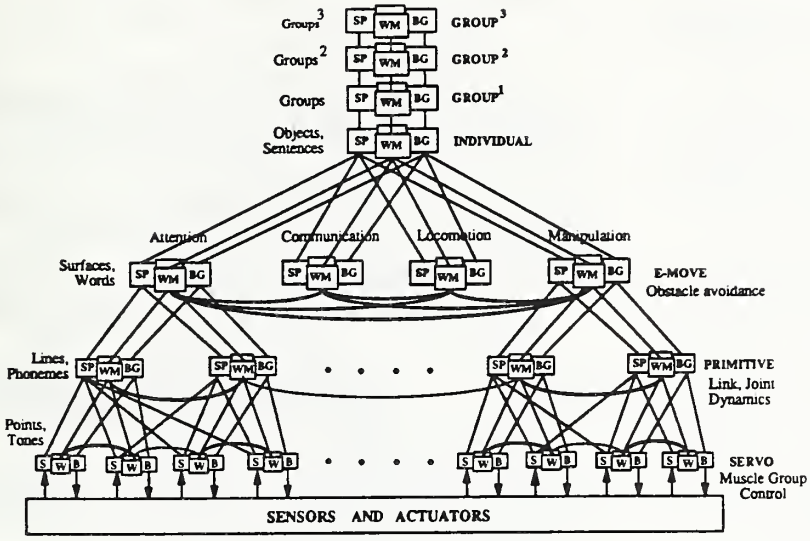



Fig. 3. An organization of processing nodes such that the BG modules form a command tree. On the right are examples or the functional characteristic of the BG modules at each level. On the left are examples of the type of visual and acoustical entities recognized by the SP modules at each level. In the center of level 3 are the type of subsystems represented by processing nodes at level 3.

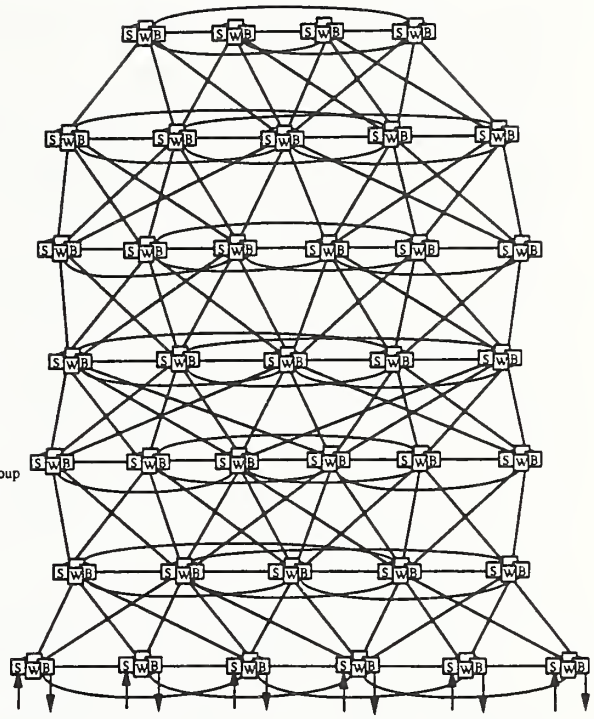


Fig. 4. Each layer of the system architecture contains a number of nodes, each of which contains BG, WM, SP, and VJ modules, The nodes are interconnected as a layered graph, or lattice, through the communication system. Note that the nodes are richly but not fully, interconnected. Outputs from the bottom layer BG modules drive actuators. Inputs to the bottom layer SP modules convey data from sensors. During operation, goal driven communication path selection mechanisms configure this lattice structure into the organization tree shown in Fig. 3.

and between nodes at the same level, especially within the same command subtree. The horizontal flow of information is most voluminous within a single node, less so between related nodes in the same command subtree, and relatively low bandwidth between computing modules in separate command subtrees. Communications bandwidth is indicated in Fig. 3 by the relative thickness of the horizontal connections.

The volume of information flowing horizontally within a subtree may be orders of magnitude larger than the amount flowing vertically in the command chain. The volume of information flowing vertically in the sensory processing system can also be very high, especially in the vision system.

The specific configuration of the command tree is task dependent, and therefore not necessarily stationary in time. Fig. 3 illustrates only one possible configuration that may exist at a single point in time. During operation, relationships between modules within and between layers of the hierarchy may be reconfigured in order to accomplish different goals, priorities, and task requirements. This means that any particular computational node, with its BG, WM, SP, and VJ modules, may belong to one subsystem at one time and a different subsystem a very short time later. For example, the mouth may be part of the manipulation subsystem (while eating) and the communication subsystem (while speaking). Similarly, an arm may be part of the manipulation subsystem (while grasping) and part of the locomotion subsystem (while swimming or climbing).

In the biological brain, command tree reconfiguration can be implemented through multiple axon pathways that exist, but are not always activated, between BG modules at different hierarchical levels. These multiple pathways define a layered graph, or lattice, of nodes and directed arcs, such as shown in Fig. 4. They enable each BG module to receive input messages and parameters from several different sources. During operation, goal driven switching mechanisms in the BG modules (discussed in Section X) assess priorities, negotiate for resources, and coordinate task activities so as to select among the possible communication paths of Fig. 4. As a result, each BG module accepts task commands from only one supervisor at a time, and hence the BG modules form a command tree at every instant in time.

The SP modules are also organized hierarchically, but as a layered graph, not a tree. At each higher level, sensory information is processed into increasingly higher levels of abstraction, but the sensory processing pathways may branch and merge in many different ways.

## VII. Hierarchical Levels

Levels in the behavior generating hierarchy are defined by temporal and spatial decomposition of goals and tasks into levels of resolution. Temporal resolution is manifested in terms of loop bandwidth, sampling rate, and state-change intervals. Temporal span is measured by the length of historical traces and planning horizons. Spatial resolution is manifested in the branching of the command tree and the resolution of maps. Spatial span is measured by the span of control and the range of maps.

Levels in the sensory processing hierarchy are defined by temporal and spatial integration of sensory data into levels of aggregation. Spatial aggregation is best illustrated by visual

images. Temporal aggregation is best illustrated by acoustic parameters such as phase, pitch, phonemes, words, sentences, rhythm, beat, and melody.

Levels in the world model hierarchy are defined by temporal resolution of events, spatial resolution of maps, and by parent-child relationships between entities in symbolic data structures. These are defined by the needs of both SP and BG modules at the various levels.

*Theorem:* In a hierarchically structured goal-driven, sensory-interactive, intelligent control system architecture:

1) control bandwidth decreases about an order of magnitude at each higher level,
2) perceptual resolution of spatial and temporal patterns decreases about an order-of-magnitude at each higher level,
3) goals expand in scope and planning horizons expand in space and time about an order-of-magnitude at each higher level, and
4) models of the world and memories of events decrease in resolution and expand in spatial and temporal range by about an order-of-magnitude at each higher level.

It is well known from control theory that hierarchically nested servo loops tend to suffer instability unless the bandwidth of the control loops differ by about an order of magnitude. This suggests, perhaps even requires, condition 1). Numerous theoretical and experimental studies support the concept of hierarchical planning and perceptual "chunking" for both temporal and spatial entities [17], [18]. These support conditions 2), 3), and 4).

In elaboration of the aforementioned theorem, we can construct a timing diagram, as shown in Fig. 5. The range of the time scale increases, and its resolution decreases, exponentially by about an order of magnitude at each higher level. Hence the planning horizon and event summary interval increases, and the loop bandwidth and frequency of subgoal events decreases, exponentially at each higher level. The seven hierarchical levels in Fig. 5 span a range of time intervals from three milliseconds to one day. Three milliseconds was arbitrarily chosen as the shortest servo update rate because that is adequate to reproduce the highest bandwidth reflex arc in the human body. One day was arbitrarily chosen as the longest historical-memory/planning-horizon to be considered. Shorter time intervals could be handled by adding another layer at the bottom. Longer time intervals could be treated by adding layers at the top, or by increasing the difference in loop bandwidths and sensory chunking intervals between levels.

The origin of the time axis in Fig. 5 is the present, i.e., $t = 0$. Future plans lie to the right of $t = 0$, past history to the left. The open triangles in the right half-plane represent task goals in a future plan. The filled triangles in the left half-plane represent recognized task-completion events in a past history. At each level there is a planning horizon and a historical event summary interval. The heavy crosshatching on the right shows the planning horizon for the current task. The light shading on the right indicates the planning horizon for the anticipated next task. The heavy crosshatching on the left shows the event summary interval for the current task. The light shading on the left shows the event summary interval for the immediately previous task.

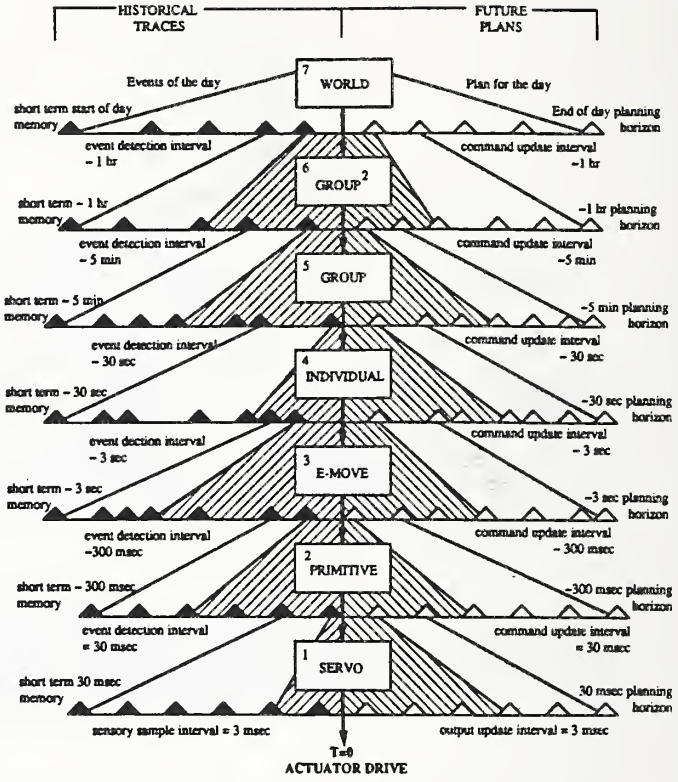



Fig. 5. Timing diagram illustrating the temporal flow of activity in the task decomposition and sensory processing systems. At the world level, high-level sensory events and circadian rhythms react with habits and daily routines to generate a plan for the day. Each elements of that plan is decomposed through the remaining six levels of task decomposition into action.

Fig. 5 suggests a duality between the behavior generation and the sensory processing hierarchies. At each hierarchical level, planner modules decompose task commands into strings of planned subtasks for execution. At each level, strings of sensed events are summarized, integrated, and "chunked" into single events at the next higher level.

Planning implies an ability to predict future states of the world. Prediction algorithms based on Fourier transforms or Kalman filters typically use recent historical data to compute parameters for extrapolating into the future. Predictions made by such methods are typically not reliable for periods longer than the historical interval over which the parameters were computed. Thus at each level, planning horizons extend into the future only about as far, and with about the same level of detail, as historical traces reach into the past.

Predicting the future state of the world often depends on assumptions as to what actions are going to be taken and what reactions are to be expected from the environment, including what actions may be taken by other intelligent agents. Planning of this type requires search over the space of possible future actions and probable reactions. Search-based planning takes place via a looping interaction between the BG, WM, and VJ modules. This is described in more detail in the Section X discussion on BG modules.

Planning complexity grows exponentially with the number

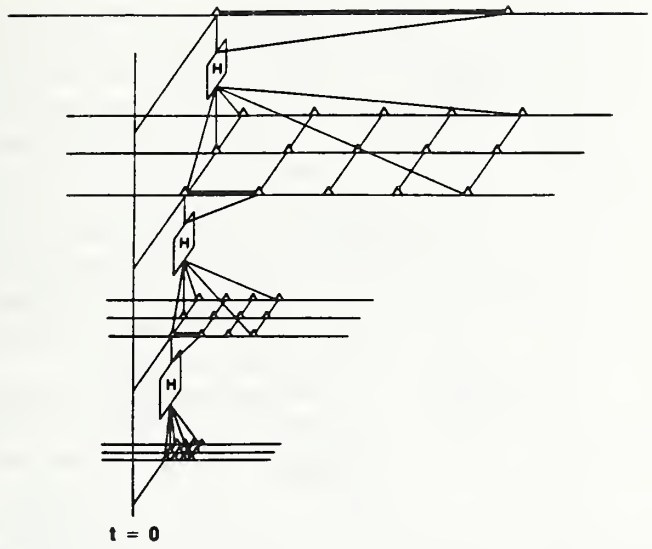



Fig. 6. Three levels of real-time planning illustrating the shrinking planning horizon and greater detail at successively lower levels of the hierarchy. At the top level, a single task is decomposed into a set of four planned subtasks for each of three subsystem. At each of the next two levels, the first task in the plan of the first subsystems is further decomposed into four subtasks for three subsystems at the next lower level.

of steps in the plan (i.e., the number of layers in the search graph). If real-time planning is to succeed, any given planner must operate in a limited search space. If there are too much resolution in the time line, or in the space of possible actions, the size of the search graph can easily become too large for real-time response. One method of resolving this problem is to use a multiplicity of planners in hierarchical layers [14], [18] so that at each layer no planner needs to search more than a given number (for example ten) steps deep in a game graph, and at each level there are no more than (ten) subsystem planners that need to simultaneously generate and coordinate plans. These criteria give rise to hierarchical levels with exponentially expanding spatial and temporal planning horizons, and characteristic degrees of detail for each level. The result of hierarchical spatiotemporal planning is illustrated in Fig. 6. At each level, plans consist of at least one, and on average 10, subtasks. The planners have a planning horizon that extends about one and a half average input command intervals into the future.

In a real-time system, plans must be regenerated periodically to cope with changing and unforeseen conditions in the world. Cyclic replanning may occur at periodic intervals. Emergency replanning begins immediately upon the detection of an emergency condition. Under full alert status, the cyclic replanning interval should be about an order of magnitude less than the planning horizon (or about equal to the expected output subtask time duration). This requires that real-time planners be able to search to the planning horizon about an order of magnitude faster than real time. This is possible only if the depth and resolution of search is limited through hierarchical planning.

Plan executors at each level have responsibility for reacting to feedback every control cycle interval. Control cycle intervals are inversely proportional to the control loop bandwidth. Typically the control cycle interval is an order of magnitude less than the expected output subtask duration. If the feedback indicates the failure of a planned subtask, the executor branches immediately (i.e., in one control cycle interval) to a preplanned emergency subtask. The planner simultaneously selects or generates an error recovery sequence that is substituted for the former plan that failed. Plan executors are also described in more detail in Section X.

When a task goal is achieved at time $t = 0$, it becomes a task completion event in the historical trace. To the extent that a historical trace is an exact duplicate of a former plan, there were no surprises; i.e., the plan was followed, and every task was accomplished as planned. To the extent that a historical trace is different from the former plan, there were surprises. The average size and frequency of surprises (i.e., differences between plans and results) is a measure of effectiveness of a planner.

At each level in the control hierarchy, the difference vector between planned (i.e., predicted) commands and observed events is an error signal, that can be used by executor submodules for servo feedback control (i.e., error correction), and by VJ modules for evaluating success and failure.

In the next eight sections, the system architecture outlined previously will be elaborated and the functionality of the computational submodules for behavior generation, world modeling, sensory processing, and value judgment will be discussed.

## VIII. Behavior Generation

*Definition:* Behavior is the result of executing a series of tasks.

*Definition:* A task is a piece of work to be done, or an activity to be performed.

*Axiom:* For any intelligent system, there exists a set of tasks that the system knows how to do.

Each task in this set can be assigned a name. The task vocabulary is the set of task names assigned to the set of tasks the system is capable of performing. For creatures capable of learning, the task vocabulary is not fixed in size. It can be expanded through learning, training, or programming. It may shrink from forgetting, or program deletion.

Typically, a task is performed by a one or more actors on one or more objects. The performance of a task can usually be described as an activity that begins with a start-event and is directed toward a goal-event. This is illustrated in Fig. 7.

*Definition:* A goal is an event that successfully terminates a task. A goal is the objective toward which task activity is directed.

*Definition:* A task command is an instruction to perform a named task. A task command may have the form: DO <Taskname(parameters)> AFTER <Start Event> UNTIL <Goal Event> Task knowledge is knowledge of how to perform a task, including information as to what tools, materials, time, resources, information, and conditions are required, plus information as to what costs, benefits and risks are expected.

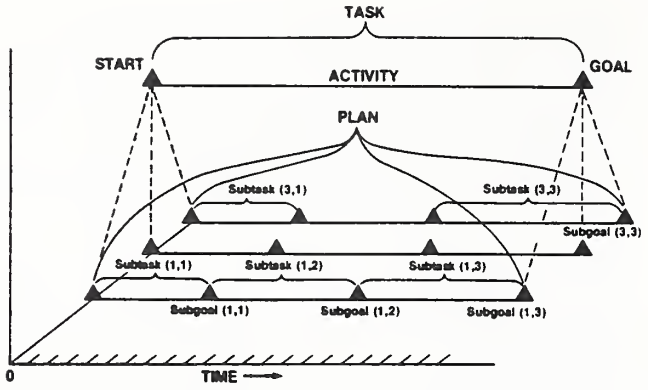



Fig. 7. A task consists of an activity that typically begins with a start event and is terminated by a goal event. A task may be decomposed into several concurrent strings of subtasks that collectively achieve the goal event.

Task knowledge may be expressed implicitly in fixed circuitry, either in the neuronal connections and synaptic weights of the brain, or in algorithms, software, and computing hardware. Task knowledge may also be expressed explicitly in data structures, either in the neuronal substrate or in a computer memory.

*Definition:* A task frame is a data structure in which task knowledge can be stored.

In systems where task knowledge is explicit, a task frame [19] can be defined for each task in the task vocabulary. An example of a task frame is:

| | |
|---|---|
| **TASKNAME** | name of the task |
| **type** | generic or specifi |
| **actor** | agent performing the task |
| **action** | activity to be performed |
| **object** | thing to be acted upon |
| **goal** | event that successfully terminates or renders the task successful |
| **parameters** | priority<br>status (e.g. active, waiting, inactive)<br>timing requirements<br>source of task command |
| **requirements** | tools, time, resources, and materials needed to perform the task<br>enabling conditions that must be satisfied to begin, or continue, the task<br>disabling conditions that will prevent, or interrupt, the task<br>information that may be required |
| **procedures** | a state-graph or state-table defining a plan for executing the task<br>functions that may be called<br>algorithms that may be needed |
| **effects** | expected results of task execution<br>expected costs, risks, benefits<br>estimated time to complete |

Explicit representation of task knowledge in task frames has a variety of uses. For example, task planners may use it for generating hypothesized actions. The world model may use it for predicting the results of hypothesized actions. The value judgment system may use it for computing how important the goal is and how many resources to expend in pursuing it. Plan executors may use it for selecting what to do next.

Task knowledge is typically difficult to discover, but once known, can be readily transferred to others. Task knowledge may be acquired by trial and error learning, but more often it is acquired from a teacher, or from written or programmed instructions. For example, the common household task of preparing a food dish is typically performed by following a recipe. A recipe is an informal task frame for cooking. Gourmet dishes rarely result from reasoning about possible combinations of ingredients, still less from random trial and error combinations of food stuffs. Exceptionally good recipes often are closely guarded secrets that, once published, can easily be understood and followed by others.

Making steel is a more complex task example. Steel making took the human race many millennia to discover how to do. However, once known, the recipe for making steel can be implemented by persons of ordinary skill and intelligence.

In most cases, the ability to successfully accomplish complex tasks is more dependent on the amount of task knowledge stored in task frames (particularly in the procedure section) than on the sophistication of planners in reasoning about tasks.

## IX. Behavior Generation

Behavior generation is inherently a hierarchical process. At each level of the behavior generation hierarchy, tasks are decomposed into subtasks that become task commands to the next lower level. At each level of a behavior generation hierarchy there exists a task vocabulary and a corresponding set of task frames. Each task frame contains a procedure state-graph. Each node in the procedure state-graph must correspond to a task name in the task vocabulary at the next lower level.

Behavior generation consists of both spatial and temporal decomposition. Spatial decomposition partitions a task into jobs to be performed by different subsystems. Spatial task decomposition results in a tree structure, where each node corresponds to a BG module, and each arc of the tree corresponds to a communication link in the chain of command as illustrated in Fig. 3.

Temporal decomposition partitions each job into sequential subtasks along the time line. The result is a set of subtasks, all of which when accomplished, achieve the task goal, as illustrated in Fig. 7.

In a plan involving concurrent job activity by different subsystems, there may requirements for coordination, or mutual constraints. For example, a start-event for a subtask activity in one subsystem may depend on the goal-event for a subtask activity in another subsystem. Some tasks may require concurrent coordinated cooperative action by several subsystems. Both planning and execution of subsystem plans may thus need to be coordinated.

There may be several alternative ways to accomplish a task. Alternative task or job decompositions can be represented by an AND/OR graph in the procedure section of the task frame. The decision as to which of several alternatives to choose is made through a series of interactions between the BG, WM, SP, and VJ modules. Each alternative may be analyzed by the BG module hypothesizing it, WM predicting the result, and VJ

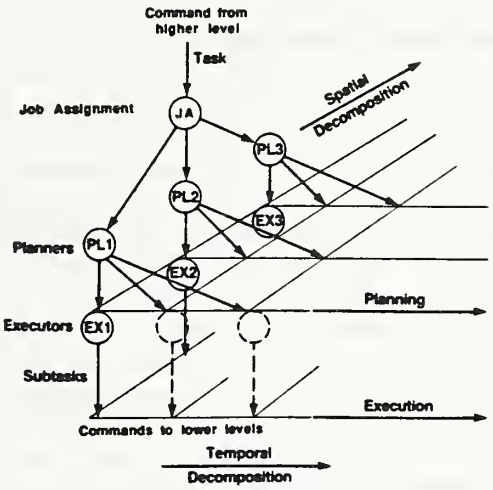



Fig. 8. The job assignment JA module performs a spatial decomposition of the task command into $N$ subsystems. For each subsystem, a planner $PL(j)$ performs a temporal decomposition of its assigned job into subtasks. For each subsystem, an executor $EX(j)$ closes a real-time control loop that servos the subtasks to the plan.

evaluating the result. The BG module then chooses the "best" alternative as the plan to be executed.

## X. BG Modules

In the control architecture defined in Fig. 3, each level of the hierarchy contains one or more BG modules. At each level, there is a BG module for each subsystem being controlled. The function of the BG modules are to decompose task commands into subtask commands.

Input to BG modules consists of commands and priorities from BG modules at the next higher level, plus evaluations from nearby VJ modules, plus information about past, present, and predicted future states of the world from nearby WM modules. Output from BG modules may consist of subtask commands to BG modules at the next lower level, plus status reports, plus "What Is?" and "What If?" queries to the WM about the current and future states of the world.

Each BG module at each level consists of three sublevels [9], [14] as shown in Fig. 8.

*The Job Assignment Sublevel—JA Submodule:* The JA submodule is responsible for spatial task decomposition. It partitions the input task command into $N$ spatially distinct jobs to be performed by $N$ physically distinct subsystems, where $N$ is the number of subsystems currently assigned to the BG module. The JA submodule many assign tools and allocate physical resources (such as arms, hands, legs, sensors, tools, and materials) to each of its subordinate subsystems for their use in performing their assigned jobs. These assignments are not necessarily static. For example, the job assignment submodule at the individual level may, at one moment, assign an arm to the manipulation subsystem in response to a <usetool> task command, and later, assign the same arm to the attention subsystem in response to a <touch/feel> task command.

The job assignment submodule selects the coordinate system in which the task decomposition at that level is to be performed. In supervisory or telerobotic control systems such as defined by NASREM [14], the JA submodule at each level may also determine the amount and kind of input to accept from a human operator.

*The Planner Sublevel—PL(j) Submodules j=1, 2, . . .N:* For each of the $N$ subsystems, there exists a planner submodule $PL(j)$. Each planner submodule is responsible for decomposing the job assigned to its subsystem into a temporal sequence of planned subtasks.

Planner submodules $PL(j)$ may be implemented by case-based planners that simply select partially or completely prefabricated plans, scripts, or schema [20]–[22] from the procedure sections of task frames. This may be done by evoking situation/action rules of the form, IF(case_$x$)/THEN(use_plan_$y$). The planner submodules may complete partial plans by providing situation dependent parameters.

The range of behavior that can be generated by a library of prefabricated plans at each hierarchical level, with each plan containing a number of conditional branches and error recovery routines, can be extremely large and complex. For example, nature has provided biological creatures with an extensive library of genetically prefabricated plans, called instinct. For most species, case-based planning using libraries of instinctive plans has proven adequate for survival and gene propagation in a hostile natural environment.

Planner submodules may also be implemented by search-based planners that search the space of possible actions. This requires the evaluation of alternative hypothetical sequences of subtasks, as illustrated in Fig. 9. Each planner $PL(j)$ hypothesizes some action or series of actions, the WM module predicts the effects of those action(s), and the VJ module computes the value of the resulting expected states of the world, as depicted in Fig. 9(a). This results in a game (or search) graph, as shown in 9(b). The path through the game graph leading to the state with the best value becomes the plan to be executed by $EX(j)$. In either case-based or search-based planning, the resulting plan may be represented by a state-graph, as shown in Fig. 9(c). Plans may also be represented by gradients, or other types of fields, on maps [23], or in configuration space.

Job commands to each planner submodule may contain constraints on time, or specify job-start and job-goal events. A job assigned to one subsystem may also require synchronization or coordination with other jobs assigned to different subsystems. These constraints and coordination requirements may be specified by, or derived from, the task frame. Each planner $PL(j)$ submodule is responsible for coordinating its plan with plans generated by each of the other $N-1$ planners at the same level, and checking to determine if there are mutually conflicting constraints. If conflicts are found, constraint relaxation algorithms [24] may be applied, or negotiations conducted between $PL(j)$ planners, until a solution is discovered. If no solution can be found, the planners report failure to the job assignment submodule, and a new job assignment may be tried, or failure may be reported to the next higher level BG module.

*The Executor Sublevel—EX(j) Submodules:* There is an executor $EX(j)$ for each planner $PL(j)$. The executor submodules are responsible for successfully executing the plan

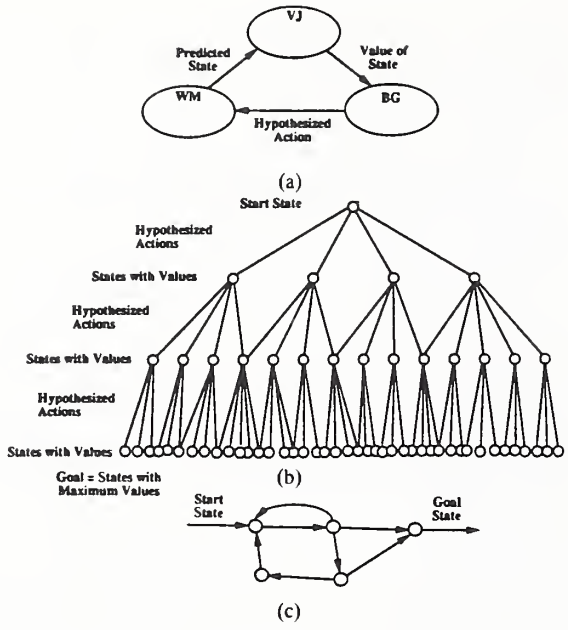



Fig. 9. Planning loop (a) produces a game graph (b). A trace in the game graph from the start to a goal state is a plan that can be represented as a plan graph (c). Nodes in the game graph correspond to edges in the plan graph, and edges in the game graph correspond to nodes in the plan graph. Multiple edges exiting nodes in the plan graph correspond to conditional branches.

state-graphs generated by their respective planners. At each tick of the state clock, each executor measures the difference between the current world state and its current plan subgoal state, and issues a subcommand designed to null the difference. When the world model indicates that a subtask in the current plan is successfully completed, the executor steps to the next subtask in that plan. When all the subtasks in the current plan are successfully executed, the executor steps to the first subtask in the next plan. If the feedback indicates the failure of a planned subtask, the executor branches immediately to a preplanned emergency subtask. Its planner meanwhile begins work selecting or generating a new plan that can be substituted for the former plan that failed. Output subcommands produced by executors at level $i$ become input commands to job assignment submodules in BG modules at level $i-1$.

Planners $PL(j)$ operate on the future. For each subsystem, there is a planner that is responsible for providing a plan that extends to the end of its planning horizon. Executors $EX(j)$ operate in the present. For each subsystem, there is an executor that is responsible for monitoring the current $(t=0)$ state of the world and executing the plan for its respective subsystem. Each executor performs a READ-COMPUTE-WRITE operation once each control cycle. At each level, each executor submodule closes a reflex arc, or servo loop. Thus, executor submodules at the various hierarchical levels form a set of nested servo loops. Executor loop bandwidths decrease on average about an order of magnitude at each higher level.

## XI. The Behavior Generating Hierarchy

Task goals and task decomposition functions often have characteristic spatial and temporal properties. For any task, there exists a hierarchy of task vocabularies that can be overlaid on the spatial/temporal hierarchy of Fig. 5.

For example:

Level 1 is where commands for coordinated velocities and forces of body components (such as arms, hands, fingers, legs, eyes, torso, and head) are decomposed into motor commands to individual actuators. Feedback servos the position, velocity, and force of individual actuators. In vertebrates, this is the level of the motor neuron and stretch reflex.

Level 2 is where commands for maneuvers of body components are decomposed into smooth coordinated dynamically efficient trajectories. Feedback servos coordinated trajectory motions. This is the level of the spinal motor centers and the cerebellum.

Level 3 is where commands to manipulation, locomotion, and attention subsystems are decomposed into collision free paths that avoid obstacles and singularities. Feedback servos movements relative to surfaces in the world. This is the level of the red nucleus, the substantia nigra, and the primary motor cortex.

Level 4 is where commands for an individual to perform simple tasks on single objects are decomposed into coordinated activity of body locomotion, manipulation, attention, and communication subsystems. Feedback initiates and sequences subsystem activity. This is the level of the basal ganglia and pre-motor frontal cortex.

Level 5 is where commands for behavior of an intelligent self individual relative to others in a small group are decomposed into interactions between the self and nearby objects or agents. Feedback initiates and steers whole self task activity. Behavior generating levels 5 and above are hypothesized to reside in temporal, frontal, and limbic cortical areas.

Level 6 is where commands for behavior of the individual relative to multiple groups are decomposed into small group interactions. Feedback steers small group interactions.

Level 7 (arbitrarily the highest level) is where long range goals are selected and plans are made for long range behavior relative to the world as a whole. Feedback steers progress toward long range goals.

The mapping of BG functionality onto levels one to four defines the control functions necessary to control a single intelligent individual in performing simple task goals. Functionality at levels one through three is more or less fixed and specific to each species of intelligent system [25]. At level 4 and above, the mapping becomes more task and situation dependent. Levels 5 and above define the control functions necessary to control the relationships of an individual relative to others in groups, multiple groups, and the world as a whole.

There is good evidence that hierarchical layers develop in the sensory-motor system, both in the individual brain as the individual matures, and in the brains of an entire species as the species evolves. It can be hypothesized that the maturation of levels in humans gives rise to Piaget's "stages of development" [26].

Of course, the biological motor system is typically much more complex than is suggested by the example model described previously. In the brains of higher species there may exist multiple hierarchies that overlap and interact with each

other in complicated ways. For example in primates, the pyramidal cells of the primary motor cortex have outputs to the motor neurons for direct control of fine manipulation as well as the inferior olive for teaching behavioral skills to the cerebellum [27]. There is also evidence for three parallel behavior generating hierarchies that have developed over three evolutionary eras [28]. Each BG module may thus contain three or more competing influences: 1) the most basic (IF it smells good, THEN eat it), 2) a more sophisticated (WAIT until the "best" moment) where best is when success probability is highest, and 3) a very sophisticated (WHAT are the long range consequences of my contemplated action, and what are all my options).

On the other hand, some motor systems may be less complex than suggested previously. Not all species have the same number of levels. Insects, for example, may have only two or three levels, while adult humans may have more than seven. In robots, the functionality required of each BG module depends upon the complexity of the subsystem being controlled. For example, one robot gripper may consist of a dexterous hand with 15 to 20 force servoed degrees of freedom. Another gripper may consist of two parallel jaws actuated by a single pneumatic cylinder. In simple systems, some BG modules (such as the Primitive level) may have no function (such as dynamic trajectory computation) to perform. In this case, the BG module will simply pass through unchanged input commands (such as <Grasp>).

## XII. The World Model

*Definition:* The world model is an intelligent system's internal representation of the external world. It is the system's best estimate of objective reality. A clear distinction between an internal representation of the world that exists in the mind, and the external world of reality, was first made in the West by Schopenhauer over 100 years ago [29]. In the East, it has been a central theme of Buddhism for millennia. Today the concept of an internal world model is crucial to an understanding of perception and cognition. The world model provides the intelligent system with the information necessary to reason about objects, space, and time. The world model contains knowledge of things that are not directly and immediately observable. It enables the system to integrate noisy and intermittent sensory input from many different sources into a single reliable representation of spatiotemporal reality.

Knowledge in an intelligent system may be represented either implicitly or explicitly. Implicit world knowledge may be embedded in the control and sensory processing algorithms and interconnections of a brain, or of a computer system. Explicit world knowledge may be represented in either natural or artificial systems by data in database structures such as maps, lists, and semantic nets. Explicit world models require computational modules capable of map transformations, indirect addressing, and list processing. Computer hardware and software techniques for implementing these types of functions are well known. Neural mechanisms with such capabilities are discussed in Section XVI.

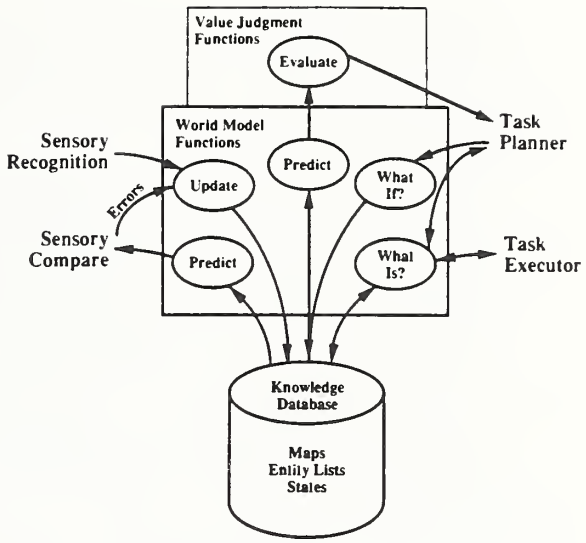



Fig. 10. Functions performed by the WM module. 1) Update knowledge database with prediction errors and recognized entities. 2) Predict sensory data. 3) Answer "What is?" queries from task executor and return current state of world. 4) Answer "What if?" queries from task planner and predict results for evaluation.

### A. WM Modules

The WM modules in each node of the organizational hierarchy of Figs. 2 and 3 perform the functions illustrated in Fig. 10.

1) WM modules maintain the knowledge database, keeping it current and consistent. In this role, the WM modules perform the functions of a database management system. They update WM state estimates based on correlations and differences between world model predictions and sensory observations at each hierarchical level. The WM modules enter newly recognized entities, states, and events into the knowledge database, and delete entities and states determined by the sensory processing modules to no longer exist in the external world. The WM modules also enter estimates, generated by the VJ modules, of the reliability of world model state variables. Believability or confidence factors are assigned to many types of state variables.
2) WM modules generate predictions of expected sensory input for use by the appropriate sensory processing SP modules. In this role, a WM module performs the functions of a signal generator, a graphics engine, or state predictor, generating predictions that enable the sensory processing system to perform correlation and predictive filtering. WM predictions are based on the state of the task and estimated states of the external world. For example in vision, a WM module may use the information in an object frame to generate real-time predicted images that can be compared pixel by pixel, or entity by entity, with observed images.
3) WM modules answer "What is?" questions asked by the planners and executors in the corresponding level BG modules. In this role, the WM modules perform the function of database query processors, question answering

systems, or data servers. World model estimates of the current state of the world are also used by BG module planners as a starting point for planning. Current state estimates are used by BG module executors for servoing and branching on conditions.

4) WM modules answer "What if?" questions asked by the planners in the corresponding level BG modules. In this role, the WM modules perform the function of simulation by generating expected status resulting from actions hypothesized by the BG planners. Results predicted by WM simulations are sent to value judgment VJ modules for evaluation. For each BG hypothesized action, a WM prediction is generated, and a VJ evaluation is returned to the BG planner. This BG-WM-VJ loop enables BG planners to select the sequence of hypothesized actions producing the best evaluation as the plan to be executed.

Data structures for representing explicit knowledge are defined to reside in a knowledge database that is hierarchically structured and distributed such that there is a knowledge database for each WM module in each node at every level of the system hierarchy. The communication system provides data transmission and switching services that make the WM modules and the knowledge database behave like a global virtual common memory in response to queries and updates from the BG, SP, and VJ modules. The communication interfaces with the WM modules in each node provides a window into the knowledge database for each of the computing modules in that node.

## XIII. Knowledge Representation

The world model knowledge database contains both *a priori* information that is available to the intelligent system before action begins, and *a posteriori* knowledge that is gained from sensing the environment as action proceeds. It contains information about space, time, entities, events, and states of the external world. The knowledge database also includes information about the intelligent system itself, such as values assigned to motives, drives, and priorities; values assigned to goals, objects, and events; parameters embedded in kinematic and dynamic models of the limbs and body; states of internal pressure, temperature, clocks, and blood chemistry or fuel level; plus the states of all of the processes currently executing in each of the BG, SP, WM, and VJ modules.

Knowledge about space is represented in maps. Knowledge about entities, events, and states is represented in lists, or frames. Knowledge about the laws of physics, chemistry, optics, and the rules of logic and mathematics are represented as parameters in the WM functions that generate predictions and simulate results of hypothetical actions. Physical knowledge may be represented as algorithms, formulae, or as IF/THEN rules of what happens under certain situations, such as when things are pushed, thrown, dropped, handled, or burned.

The correctness and consistency of world model knowledge is verified by sensory processing mechanisms that measure differences between world model predictions and sensory observations.

### A. Geometrical Space

From psychophysical evidence Gibson [30] concludes that the perception of geometrical space is primarily in terms of "medium, substance, and the surfaces that separate them". Medium is the air, water, fog, smoke, or falling snow through which the world is viewed. Substance is the material, such as earth, rock, wood, metal, flesh, grass, clouds, or water, that comprise the interior of objects. The surfaces that separate the viewing medium from the viewed objects is what are observed by the sensory system. The sensory input thus describes the external physical world primarily in terms of surfaces.

Surfaces are thus selected as the fundamental element for representing space in the proposed WM knowledge database. Volumes are treated as regions between surfaces. Objects are defined as circumscribed, often closed, surfaces. Lines, points and vertices lie on, and may define surfaces. Spatial relationships on surfaces are represented by maps.

### B. Maps

*Definition:* A map is a two dimensional database that defines a mesh or grid on a surface.

The surface represented by a map may be, but need not be, flat. For example, a map may be defined on a surface that is draped over, or even wrapped around, a three-dimensional (3–D) volume.

*Theorem:* Maps can be used to describe the distribution of entities in space.

It is always possible and often useful to project the physical 3-D world onto a 2-D surface defined by a map. For example, most commonly used maps are produced by projecting the world onto the 2-D surface of a flat sheet of paper, or the surface of a globe. One great advantage of such a projection is that it reduces the dimensionality of the world from three to two. This produces an enormous saving in the amount of memory required for a database representing space. The saving may be as much as three orders of magnitude, or more, depending on the resolution along the projected dimension.

*1) Map Overlays:* Most of the useful information lost in the projection from 3-D space to a 2-D surface can be recovered through the use of map overlays.

*Definition:* A map overlay is an assignment of values, or parameters, to points on the map.

A map overlay can represent spatial relationships between 3-D objects. For example, an object overlay may indicate the presence of buildings, roads, bridges, and landmarks at various places on the map. Objects that appear smaller than a pixel on a map can be represented as icons. Larger objects may be represented by labeled regions that are projections of the 3-D objects on the 2-D map. Objects appearing on the map overlay may be cross referenced to an object frame database elsewhere in the world model. Information about the 3-D geometry of objects on the map may be represented in the object frame database.

Map overlays can also indicate attributes associated with points (or pixels) on the map. One of the most common map overlays defines terrain elevation. A value of terrain elevation

$(z)$ overlaid at each $(x, y)$ point on a world map produces a topographic map.

A map can have any number of overlays. Map overlays may indicate brightness, color, temperature, even "behind" or "in-front". A brightness or color overlay may correspond to a visual image. For example, when aerial photos or satellite images are registered with map coordinates, they become brightness or color map overlays.

Map overlays may indicate terrain type, or region names, or can indicate values, such as cost or risk, associated with regions. Map overlays can indicate which points on the ground are visible from a given location in space. Overlays may also indicate contour lines and grid lines such as latitude and longitude, or range and bearing.

Map overlays may be useful for a variety of functions. For example, terrain elevation and other characteristics may be useful for route planning in tasks of manipulation and locomotion. Object overlays can be useful for analyzing scenes and recognizing objects and places.

A map typically represents the configuration of the world at a single instant in time, i.e., a snapshot. Motion can be represented by overlays of state variables such as velocity or image flow vectors, or traces (i.e., trajectories) of entity locations. Time may be represented explicitly by a numerical parameter associated with each trajectory point, or implicitly by causing trajectory points to fade, or be deleted, as time passes.

*Definition:* A map pixel frame is a frame that contains attributes and attribute-values attached to that map pixel.

*Theorem:* A set of map overlays are equivalent to a set of map pixel frames.

*Proof:* If each map overlay defines a parameter value for every map pixel, then the set of all overlay parameter values for each map pixel defines a frame for that pixel. Conversely, the frame for each pixel describes the region covered by that pixel. The set of all pixel frames thus defines a set of map overlays, one overlay for each attribute in the pixel frames. Q.E.D.

For example, a pixel frame may describe the color, range, and orientation of the surface covered by the pixel. It may describe the name of (or pointer to) the entities to which the surface covered by the pixel belongs. It may also contain the location, or address, of the region covered by the pixel in other coordinate systems.

In the case of a video image, a map pixel frame might have the following form:

| | |
|---|---|
| **PIXEL_NAME** | $(AZ.EL)$ location index on map (Sensor egosphere coordinates) |
| **brightness** | $I$ |
| **color** | $I_r.I_b.I_g$ |
| **spatial brightness gradient** | $dI/dAZ.\ dI/dEL$ (sensor egosphere coordinates) |
| **temporal brightness gradient** | $dI/dt$ |
| **image flow direction** | $B$ (velocity egosphere coordinates) |
| **image flow rate** | $dA/dt$ (velocity egosphere coordinates) |
| **range** | $R$ to surface covered (from egosphere origin) |
| **head egosphere location** | $az$, $el$ of egosphere ray to surface covered |
| **world map location** | $x.y.z$ of map point on surface covered |
| **world map location** | $x.y.z$ of map point on surface covered |
| **linear feature pointer** | pointer to frame of line, edge, or vertex covered by pixel |
| **surface feature pointer** | pointer to frame of surface covered by pixel |
| **object pointer** | pointer to frame of object covered by pixel |
| **object map location** | $X.Y.Z$ of surface covered in object coordinates group pointer pointer to group covered by pixel |

Indirect addressing through pixel frame pointers can allow value state-variables assigned to objects or situations to be inherited by map pixels. For example, value state-variables such as attraction-repulsion, love-hate, fear-comfort assigned to objects and map regions can also be assigned through inheritance to individual map and egosphere pixels.

There is some experimental evidence to suggest that map pixel frames exist in the mammalian visual system. For example, neuron firing rates in visual cortex have been observed to represent the values of attributes such as edge orientation, edge and vertex type, and motion parameters such as velocity, rotation, and flow field divergence. These firing rates are observed to be registered with retinotopic brightness images [31], [54].

### C. Map Resolution

The resolution required for a world model map depends on how the map is generated and how it is used. All overlays need not have the same resolution. For predicting sensory input, world model maps should have resolution comparable to the resolution of the sensory system. For vision, map resolution may be on the order of 64K to a million pixels. This corresponds to image arrays of $256 \times 256$ pixels to $1000 \times 1000$ pixels respectively. For other sensory modalities, resolution can be considerably less.

For planning, different levels of the control hierarchy require maps of different scale. At higher levels, plans cover long distances and times, and require maps of large area, but low resolution. At lower levels, plans cover short distances and times, and maps need to cover small areas with high resolution. [18]

World model maps generated solely from symbolic data in long term memory may have resolution on the order of a few thousand pixels or less. For example, few humans can recall from memory the relative spatial distribution of as many as a hundred objects, even in familiar locations such as their own homes. The long term spatial memory of an intelligent creature typically consists of a finite number of relatively small regions that may be widely separated in space; for example,

one's own home, the office, or school, the homes of friends and relatives, etc. These known regions are typically connected by linear pathways that contain at most a few hundred known waypoints and branchpoints. The remainder of the world is known little, or not at all. Unknown regions, which make up the vast majority of the real world, occupy little or no space in the world model.

The efficient storage of maps with extremely nonuniform resolution can be accomplished in a computer database by quadtrees [32], hash coding, or other sparse memory representations [33]. Pathways between known areas can be economically represented by graph structures either in neuronal or electronic memories. Neural net input-space representations and transformations such as are embodied in a CMAC [34], [35] give insight as to how nonuniformly dense spatial information might be represented in the brain.

### D. Maps and Egospheres

It is well known that neurons in the brain, particularly in the cortex, are organized as 2-D arrays, or maps. It is also known that conformal mappings of image arrays exist between the retina, the lateral geniculate, the superior colliculus, and several cortical visual areas. Similar mappings exist in the auditory and tactile sensory systems. For every map, there exists a coordinate system, and each map pixel has coordinate values. On the sensor egosphere, pixel coordinates are defined by the physical position of the pixel in the sensor array. The position of each pixel in other map coordinate systems can be defined either by neuronal interconnections, or by transform parameters contained in each pixel's frame.

There are three general types of map coordinate systems that are important to an intelligent system: world coordinates, object coordinates, and egospheres.

*1) World Coordinates:* World coordinate maps are typically flat 2-D arrays that are projections of the surface of the earth along the local vertical. World coordinates are often expressed in a Cartesian frame, and referenced to a point in the world. In most cases, the origin is an arbitrary point on the ground. The $z$ axis is defined by the vertical, and the $x$ and $y$ axes define points on the horizon. For example, $y$ may point North and $x$ East. The value of $z$ is often set to zero at sea level.

World coordinates may also be referenced to a moving point in the world. For example, the origin may be the self, or some moving object in the world. In this case, stationary pixels on the world map must be scrolled as the reference point moves.

There may be several world maps with different resolutions and ranges. These will be discussed near the end of this section.

*2) Object Coordinates:* Object coordinates are defined with respect to features in an object. For example, the origin might be defined as the center of gravity, with the coordinate axes defined by axes of symmetry, faces, edges, vertices, or skeletons [36]. There are a variety of surface representations that have been suggested for representing object geometry. Among these are generalized cylinders [37], [38], B-splines [39], quadtrees [32], and aspect graphs [40]. Object coordinate maps are typically 2-D arrays of points painted on the surfaces of objects in the form of a grid or mesh. Other boundary representation can usually be transformed into this form.

Object map overlays can indicate surface characteristics such as texture, color, hardness, temperature, and type of material. Overlays can be provided for edges, boundaries, surface normal vectors, vertices, and pointers to object frames containing center lines, centroids, moments, and axes of symmetry.

*3) Egospheres:* An egosphere is a two–dimensional (2-D) spherical surface that is a map of the world as seen by an observer at the center of the sphere. Visible points on regions or objects in the world are projected on the egosphere wherever the line of sight from a sensor at the center of the egosphere to the points in the world intersect the surface of the sphere. Egosphere coordinates thus are polar coordinates defined by the self at the origin. As the self moves, the projection of the world flows across the surface of the egosphere.

Just as the world map is a flat 2-D $(x, y)$ array with multiple overlays, so the egosphere is a spherical 2-D $(AZ.EL)$ array with multiple overlays. Egosphere overlays can attribute brightness, color, range, image flow, texture, and other properties to regions and entities on the egosphere. Regions on the egosphere can thus be segmented by attributes, and egosphere points with the same attribute value may be connected by contour lines. Egosphere overlays may also indicate the trace, or history, of brightness values or entity positions over some time interval. Objects may be represented on the egosphere by icons, and each object may have in its database frame a trace, or trajectory, of positions on the egosphere over some time interval.

### E. Map Transformations

*Theorem:* If surfaces in real world space can be covered by an array (or map) of points in a coordinate system defined in the world, and the surface of a WM egosphere is also represented as an array of points, then there exists a function $G$ that transforms each point on the real world map into a point on the WM egosphere, and a function $G'$ that transforms each point on the WM egosphere for which range is known into a point on the real world map.

*Proof:* Fig. 11 shows the 3-D relationship between an egosphere and world map coordinates. For every point $(x.y,z)$ in world coordinates, there is a point $(AZ.EL.R)$ in ego centered coordinates that can be computed by the $3 \times 3$ matrix function $G$

$$(AZ.EL,R)^T = G(x.y,z)^T$$

There, of course, may be more than one point in the world map that gives the same $(AZ.EL)$ values on the egosphere. Only the $(AZ.EL)$ with the smallest value of $R$ will be visible to an observer at the center of the egosphere. The deletion of egosphere pixels with $R$ larger than the smallest for each value of $(AZ,EL)$ corresponds to the hidden surface removal problem common in computer graphics.

For each egosphere pixel where $R$ is known, $(x.y.z)$ can be computed from $(AZ,EL,R)$ by the function $G'$

$$(x,y,z)^T = G'(AZ.EL.R)^T$$

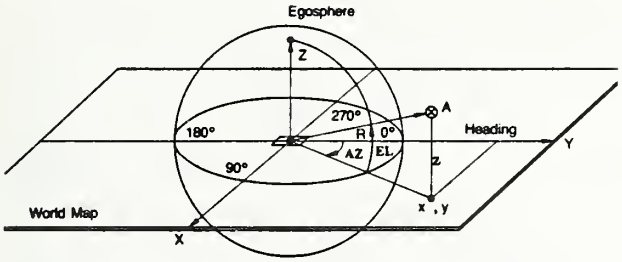



Fig. 11. Geometric relationship between world map and egosphere coordinates.

Any point in the world topological map can thus be projected onto the egosphere (and vice versa when $R$ is known). Projections from the egosphere to the world map will leave blank those map pixels that cannot be observed from the center of the egosphere. Q.E.D.

There are 2 × 2 transformations of the form

$$(AZ.EL)^T = F(az.el)^T$$

and

$$(az,el)^T = F'(AZ.EL)^T$$

that can relate any map point $(AZ.EL)$ on one egosphere to a map point (az,el) on another egosphere of the same origin. The radius $R$ to any egosphere pixel is unchanged by the $F$ and $F'$ transformations between egosphere representations with the same origin.

As ego motion occurs (i.e., as the self object moves through the world), the egosphere moves relative to world coordinates, and points on the egocentric maps flow across their surfaces. Ego motion may involve translation, or rotation, or both; in a stationary world, or a world containing moving objects. If egomotion is known, range to all stationary points in the world can be computed from observed image flow; and once range to any stationary point in the world is known, its pixel motion on the egosphere can be predicted from knowledge of egomotion. For moving points, prediction of pixel motion on the egosphere requires additional knowledge of object motion.

### *F. Egosphere Coordinate Systems*

The proposed world model contains four different types of egosphere coordinates:

*1) Sensor Egosphere Coordinates:* The sensor egosphere is defined by the sensor position and orientation, and moves as the sensor moves. For vision, the sensor egosphere is the coordinate system of the retina. The sensor egosphere has coordinates of azimuth $(AZ)$ and elevation $(EL)$ fixed in the sensor system (such as an eye or a TV camera), as shown in Fig. 12. For a narrow field of view, rows and columns $(x,z)$ in a flat camera image array correspond quite closely to azimuth and elevation $(AZ,EL)$ on the sensor egosphere. However, for a wide field of view, the egosphere and flat image array representations have widely different geometries. The flat image $(x,z)$ representation becomes highly elongated for a wide field of view, going to infinity at plus and minus 90 degrees. The egosphere representation, in contrast, is well behaved over the entire sphere (except for singularities at the egosphere poles).

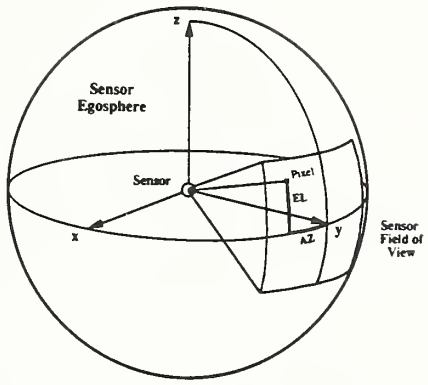



Fig. 12. Sensor egosphere coordinates. Azimuth (AZ) is measured clockwise from the sensor $y$-axis in the $x-y$ plane. Elevation (EL) is measured up and down (plus and minus) from the $x-y$ plane.

The sensor egosphere representation is useful for the analysis of wide angle vision such as occurs in the eyes of most biological creatures. For example, most insects and fish, many birds, and most prey animals such as rabbits have eyes with fields of view up to 180 degrees. Such eyes are often positioned on opposite sides of the head so as to provide almost 360 degree visual coverage. The sensor egosphere representation provides a tractable coordinate frame in which this type of vision can be analyzed.

*2) Head Egosphere Coordinates:* The head egosphere has $(AZ,EL)$ coordinates measured in a reference frame fixed in the head (or sensor platform). The head egosphere representation is well suited for fusing sensory data from multiple sensors, each of which has its own coordinate system. Vision data from multiple eyes or cameras can be overlaid and registered in order to compute range from stereo. Directional and range data from acoustic and sonar sensors can be overlaid on vision data. Data derived from different sensors, or from multiple readings of the same sensor, can be overlaid on the head egosphere to build up a single image of multidimensional reality.

Pixel data in sensor egosphere coordinates can be transformed into the head egosphere by knowledge of the position and orientation of the sensor relative to the head. For example, the position of each eye in the head is fixed and the orientation of each eye relative to the head is known from stretch sensors in the ocular muscles. The position of tactile sensors relative to the head is known from proprioceptive sensors in the neck, torso, and limbs.

*Hypothesis:* Neuronal maps on the tectum (or superior colliculus), and on parts of the extrastriate visual cortex, are represented in a head egosphere coordinate system.

Receptive fields from the two retinas are well known to be overlaid in registration on the tectum, and superior colliculus. Experimental evidence indicates that registration and fusion of data from visual and auditory sensors takes place in the tectum of the barn owl [41] and the superior colliculus of the monkey [42] in head egosphere coordinates. Motor output for eye motion from the superior colliculus apparently is transformed

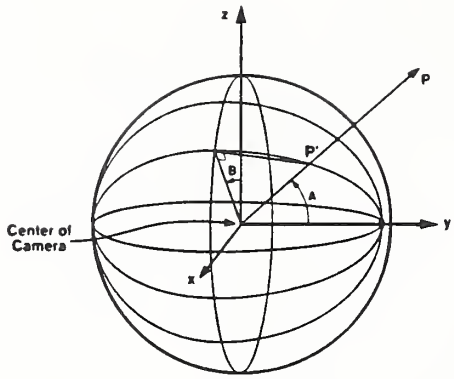



Fig. 13. The velocity egosphere. On the velocity egosphere, the $y$-axis is defined by the velocity factor. the $x$-axis points to the horizon on the right. $A$ is the angle between the velocity vector and a pixel on the egosphere, and $B$ is the angles between the $z$-axis and the plane defined by the velocity vector and the pixel vector.

back into retinal egosphere coordinates. There is also evidence that head egosphere coordinates are used in the visual areas of the parietal cortex [43], [54].

*3) Velocity Egosphere:* The velocity egosphere is defined by the velocity vector and the horizon. The velocity vector defines the pole ($y$-axis) of the velocity egosphere, and the x-axis points to the right horizon as shown in Fig. 13. The egosphere coordinates $(A, B)$ are defined such that $A$ is the angle between the pole and a pixel, and $B$ is the angle between the $yoz$ plane and the plane of the great circle flow line containing the pixel.

For egocenter translation without rotation through a stationary world, image flow occurs entirely along great circle arcs defined by $B$ =constant. The positive pole of the velocity egosphere thus corresponds to the focus-of-expansion. The negative pole corresponds to the focus-of-contraction. The velocity egosphere is ideally suited for computing range from image flow, as discussed in Section XIV.

*4) Inertial Egosphere:* The inertial egosphere has coordinates of azimuth measured from a fixed point (such as North) on the horizon, and elevation measured from the horizon.

The inertial egosphere does not rotate as a result of sensor or body rotation. On the inertial egosphere, the world is perceived as stationary despite image motion due to rotation of the sensors and the head.

Fig. 14 illustrates the relationships between the four egosphere coordinate systems. Pixel data in eye (or camera) egosphere coordinates can be transformed into head (or sensor platform) egosphere coordinates by knowledge of the position and orientation of the sensor relative to the head. For example, the position of each eye in the head is fixed and the orientation of each eye relative to the head is known from stretch receptors in the ocular muscles (or pan and tilt encoders on a camera platform). Pixel data in head egosphere coordinates can be transformed into inertial egosphere coordinates by knowing the orientation of the head in inertial space. This information can be obtained from the vestibular (or inertial) system that measures the direction of gravity relative to the head and integrates rotary accelerations to obtain head position in inertial space. The inertial egosphere can be transformed

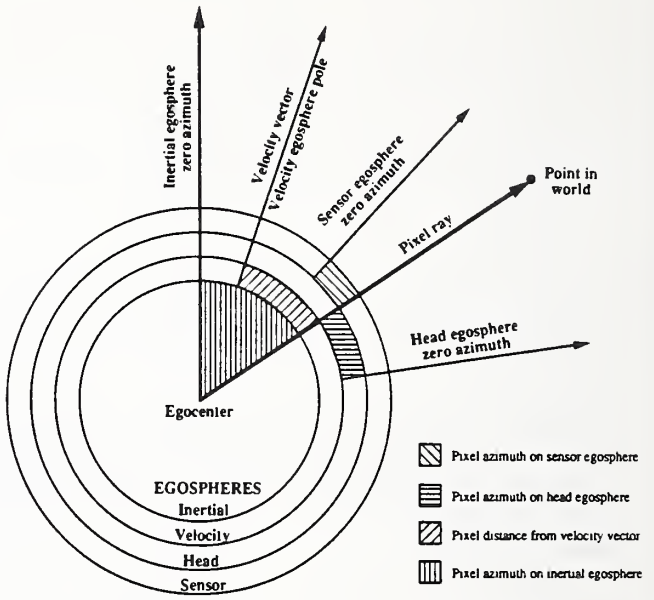



Fig. 14. A 2-D projection of four egosphere representations illustrating angular relationships between egospheres. Pixels are represented on each egosphere such that images remains in registration. Pixel attributes detected on one egosphere may thus be inherited on others. Pixel resolution is not typically uniform on a single egosphere, nor is it necessarily the same for different egospheres, or for different attributes on the same egosphere.

into world coordinates by knowing the $x, y, z$ position of the center of the egosphere. This is obtained from knowledge about where the self is located in the world. Pixels on any egosphere can be transformed into the velocity egosphere by knowledge of the direction of the current velocity vector on that egosphere. This can be obtained from a number of sources including the locomotion and vestibular systems.

All of the previous egosphere transformations can be inverted, so that conversions can be made in either direction. Each transformation consists of a relatively simple vector function that can be computed for each pixel in parallel. Thus the overlay of sensory input with world model data can be accomplished in a few milliseconds by the type of computing architectures known to exist in the brain. In artificial systems, full image egosphere transformations can be accomplished within a television frame interval by state-of-the-art serial computing hardware. Image egosphere transformations can be accomplished in a millisecond or less by parallel hardware.

*Hypothesis:* The WM world maps, object maps, and egospheres are the brains data fusion mechanisms. They provide coordinate systems in which to integrate information from arrays of sensors (i.e., rods and cones in the eyes, tactile sensors in the skin, directional hearing, etc.) in space and time. They allow information from different sensory modalities (i.e., vision, hearing, touch, balance, and proprioception) to be combined into a single consistent model of the world.

*Hypothesis:* The WM functions that transform data between the world map and the various egosphere representations are the brain's geometry engine. They transform world model predictions into the proper coordinate systems for real-time comparison and correlation with sensory observations. This provides the basis for recognition and perception.

Transformations to and from the sensor egosphere, the inertial egosphere, the velocity egosphere, and the world map allow the intelligent system to sense the world from one perspective and interpret it in another. They allow the intelligent system to compute how entities in the world would look from another viewpoint. They provide the ability to overlay sensory input with world model predictions, and to compute the geometrical and dynamical functions necessary to navigate, focus attention, and direct action relative to entities and regions of the world.

### G. Entities

*Definition:* An entity is an element from the set {point, line, surface, object, group}.

The world model contains information about entities stored in lists, or frames. The knowledge database contains a list of all the entities that the intelligent system knows about. A subset of this list is the set of current-entities known to be present in any given situation. A subset of the list of current-entities is the set of entities-of-attention.

There are two types of entities: generic and specific. A generic entity is an example of a class of entities. A generic entity frame contains the attributes of its class. A specific entity is a particular instance of an entity. A specific entity frame inherits the attributes of the class to which it belongs. An example of an entity frame might be:

| | |
|---|---|
| **ENTITY NAME** | name of entity |
| **kind** | class or species of entity |
| **type** | generic or specific point, line, surface, object, or group |
| **position** | world map coordinates (uncertainty); egosphere coordinates (uncertainty) |
| **dynamics** | velocity (uncertainty);acceleration (uncertainty) |
| **trajectory** | sequence of positions |
| **geometry** | center of gravity (uncertainty); axis of symmetry (uncertainty);size (uncertainty);shape boundaries (uncertainty) |
| **links** | subentities; parent entity |
| **properties** | physical: mass; color; substance; behavioral: social (of animate objects) |
| **capabilities** | speed, range |
| **value state-variables** | attract-repulse; confidence-fear; love-hate |

For example, upon observing a specific cow named Bertha, an entity frame in the brain of a visitor to a farm might have the following values:

| | |
|---|---|
| **ENTITY NAME** | Bertha |
| **kind** | cow |
| **type** | specific object |
| **position** | $x, y, z$ (in pasture map coordinates) $AZ, EL, R$ (in egosphere image of observer) |
| **dynamics** | velocity, acceleration (in egosphere or pasture map coordinates) |
| **trajectory** | sequence of map positions while grazing |
| **geometry** | axis of symmetry (right/left) size (6 × 3 × 10 ft) shape (quadruped) |
| **links** | subentities - surfaces (torso, neck, head, legs, tail, etc.) parent entity - group (herd) |
| **properties** | physical:mass (1050 lbs); color (black and white); substance (flesh, bone, skin, hair); behavioral (standing, placid, timid, etc.) |
| **capabilities** | speed, range |
| **value state-variables** | attract-repulse = 3 (visitor finds cows moderately attractive) confidence-fear= -2 (visitor slightly afraid of cows) love-hate = 1 (no strong feelings) |

### H. Map–Entity Relationship

Map and entity representations are cross referenced and tightly coupled by real-time computing hardware. Each pixel on the map has in its frame a pointer to the list of entities covered by that pixel. For example, each pixel may cover a point entity indicating brightness, color, spatial and temporal gradients of brightness and color, image flow, and range for each point. Each pixel may also cover a linear entity indicating a brightness or depth edge or vertex; a surface entity indicating area, slope, and texture; an object entity indicating the name and attributes of the object covered; a group entity indicating the name and attributes of the group covered, etc.

Likewise, each entity in the attention list may have in its frame a set of geometrical parameters that enables the world model geometry engine to compute the set of egosphere or world map pixels covered by each entity, so that entity parameters associated with each pixel covered can be overlaid on the world and egosphere maps.

Cross referencing between pixel maps and entity frames allows the results of each level of processing to add map overlays to the egosphere and world map representations. The entity database can be updated from knowledge of image parameters at points on the egosphere, and the map database can be predicted from knowledge of entity parameters in the world model. At each level, local entity and map parameters can be computed in parallel by the type of neurological computing structures known to exist in the brain.

Many of the attributes in an entity frame are time dependent state-variables. Each time dependent state-variable may possess a short term memory queue wherein is stored a state trajectory, or trace, that describes its temporal history.

At each hierarchical level, temporal traces stretch backward about as far as the planning horizon at that level stretches into the future. At each hierarchical level, the historical trace of an entity state-variable may be captured by summarizing data values at several points in time throughout the historical interval. Time dependent entity state-variable histories may also be captured by running averages and moments, Fourier transform coefficients, Kalman filter parameters, or other analogous methods.

Each state-variable in an entity frame may have value state-variable parameters that indicate levels of believability, confidence, support, or plausibility, and measures of dimensional uncertainty. These are computed by value judgment functions that reside in the VJ modules. These are described in Section XV.

Value state-variable parameters may be overlaid on the map and egosphere regions where the entities to which they are assigned appear. This facilitates planning. For example, approach-avoidance behavior can be planned on an egosphere map overlay defined by the summation of attractor and repulsor value state-variables assigned to objects or regions that appear on the egosphere. Navigation planning can be done on a map overlay whereon risk and benefit values are assigned to regions on the egosphere or world map.

### *I. Entity Database Hierarchy*

The entity database is hierarchically structured. Each entity consists of a set of subentities, and is part of a parent entity. For example, an object may consist of a set of surfaces, and be part of a group.

The definition of an object is quite arbitrary, however, at least from the point of view of the world model. For example, is a nose an object? If so, what is a face? Is a head an object? Or is it part of a group of objects comprising a body? If a body can be a group, what is a group of bodies?

Only in the context of a task, does the definition of an object become clear. For example, in a task frame, an object may be defined either as the agent, or as acted upon by the agent executing the task. Thus, in the context of a specific task, the nose (or face, or head) may become an object because it appears in a task frame as the agent or object of a task.

Perception in an intelligent system is task (or goal) driven, and the structure of the world model entity database is defined by, and may be reconfigured by, the nature of goals and tasks. It is therefore not necessarily the role of the world model to define the boundaries of entities, but rather to represent the boundaries defined by the task frame, and to map regions and entities circumscribed by those boundaries with sufficient resolution to accomplish the task. It is the role of the sensory processing system to identify regions and entities in the external real world that correspond to those represented in the world model, and to discover boundaries that circumscribe objects defined by tasks.

*Theorem:* The world model is hierarchically structured with map (iconic) and entity (symbolic) data structures at each level of the hierarchy.

At level 1, the world model can represent map overlays for point entities. In the case of vision, point entities may consist of brightness or color intensities, and spatial and temporal derivatives of those intensities. Point entity frames include brightness spatial and temporal gradients and range from stereo for each pixel. Point entity frames also include transform parameters to and from head egosphere coordinates. These representations are roughly analogous to Marr's "primal sketch" [44], and are compatible with experimentally observed data representations in the tectum, superior colliculus, and primary visual cortex ($V1$) [31].

At level 2, the world model can represent map overlays for linear entities consisting of clusters, or strings, of point entities. In the visual system, linear entities may consist of connected edges (brightness, color, or depth), vertices, image flow vectors, and trajectories of points in space/time. Attributes such as 3-D position, orientation, velocity, and rotation are represented in a frame for each linear entity. Entity frames include transform parameters to and from inertial egosphere coordinates. These representations are compatible with experimentally observed data representations in the secondary visual cortex (V2) [54].

At level 3, the world model can represent map overlays for surface entities computed from sets of linear entities clustered or swept into bounded surfaces or maps, such as terrain maps, B-spline surfaces, or general functions of two variables. Surface entities frames contain transform parameters to and from object coordinates. In the case of vision, entity attributes may describe surface color, texture, surface position and orientation, velocity, size, rate of growth in size, shape, and surface discontinuities or boundaries. Level 3 is thus roughly analogous to Marr's "2 1/2-D sketch", and is compatible with known representation of data in visual cortical areas V3 and V4.

At level 4, the world model can represent map overlays for object entities computed from sets of surfaces clustered or swept so as to define 3-D volumes, or objects. Object entity frames contain transform parameters to and from object coordinates. Object entity frames may also represent object type, position, translation, rotation, geometrical dimensions, surface properties, occluding objects, contours, axes of symmetry, volumes, etc. These are analogous to Marr's "3-D model" representation, and compatible with data representations in occipital-temporal and occipital-parietal visual areas.

At level 5, the world model can represent map overlays for group entities consisting of sets of objects clustered into groups or packs. This is hypothesized to correspond to data representations in visual association areas of parietal and temporal cortex. Group entity frames contain transform parameters to and from world coordinates. Group entity frames may also represent group species, center of mass, density, motion. map position, geometrical dimensions, shape, spatial axes of symmetry, volumes, etc.

At level 6, the world model can represent map overlays for sets of group entities clustered into groups of groups, or group$^2$ entities. At level 7, the world model can represent map overlays for sets of group$^2$ entities clustered into group$^3$ (or world) entities, and so on. At each higher level, world map resolution decreases and range increases by about an order of

magnitude per level.

The highest level entity in the world model is the world itself, i.e., the environment as a whole. The environment entity frame contains attribute state-variables that describe the state of the environment, such as temperature, wind, precipitation, illumination, visibility, the state of hostilities or peace, the current level of danger or security, the disposition of the gods, etc.

### J. Events

*Definition:* An event is a state, condition, or situation that exists at a point in time, or occurs over an interval in time.

Events may be represented in the world model by frames with attributes such as the point, or interval, in time and space when the event occurred, or is expected to occur. Event frames attributes may indicate start and end time, duration, type, relationship to other events, etc.

An example of an event frame is:

| | |
|---|---|
| **EVENT NAME** | name of event |
| **kind** | class or species |
| **type** | generic or specific |
| **modality** | visual, auditory, tactile, etc. |
| **time** | when event detected |
| **interval** | period over which event took place |
| **position** | map location where event occurred |
| **links** | subevents; parent event |
| **value** | good-bad, benefit-cost, etc. |

State-variables in the event frame may have confidence levels, degrees of support and plausibility, and measures of dimensional uncertainty similar to those in spatial entity frames. Confidence state-variables may indicate the degree of certainty that an event actually occurred, or was correctly recognized.

The event frame database is hierarchical. At each level of the sensory processing hierarchy, the recognition of a pattern, or string, of level($i$) events makes up a single level($i+1$) event.

*Hypothesis:* The hierarchical levels of the event frame database can be placed in one-to-one correspondence with the hierarchical levels of task decomposition and sensory processing.

For example at: Level 1—an event may span a few milliseconds. A typical level(1) acoustic event might be the recognition of a tone, hiss, click, or a phase comparison indicating the direction of arrival of a sound. A typical visual event might be a change in pixel intensity, or a measurement of brightness gradient at a pixel.

Level 2—an event may span a few tenths of a second. A typical level(2) acoustic event might be the recognition of a phoneme or a chord. A visual event might be a measurement of image flow or a trajectory segment of a visual point or feature.

Level 3—an event may span a few seconds, and consist of the recognition of a word, a short phrase, or a visual gesture, or motion of a visual surface.

Level 4—an event may span a few tens of seconds, and consist of the recognition of a message, a melody, or a visual observation of object motion, or task activity.

Level 5—an event may span a few minutes and consist of listening to a conversation, a song, or visual observation of group activity in an extended social exchange.

Level 6—an event may span an hour and include many auditory, tactile, and visual observations.

Level 7—an event may span a day and include a summary of sensory observations over an entire day's activities.

## XIV. Sensory Processing

*Definition:* Sensory processing is the mechanism of perception.

*Theorem:* Perception is the establishment and maintenance of correspondence between the internal world model and the external real world.

*Corollary:* The function of sensory processing is to extract information about entities, events, states, and relationships in the external world, so as keep the world model accurate and up to date.

### A. Measurement of Surfaces

World model maps are updated by sensory measurement of points, edges, and surfaces. Such information is usually derived from vision or touch sensors, although some intelligent systems may derive it from sonar, radar, or laser sensors.

The most direct method of measuring points, edges, and surfaces is through touch. Many creatures, from insects to mammals, have antennae or whiskers that are used to measure the position of points and orientation of surfaces in the environment. Virtually all creatures have tactile sensors in the skin, particularly in the digits, lips, and tongue. Proprioceptive sensors indicate the position of the feeler or tactile sensor relative to the self when contact is made with an external surface. This, combined with knowledge of the kinematic position of the feeler endpoint, provides the information necessary to compute the position on the egosphere of each point contacted. A series of felt points defines edges and surfaces on the egosphere.

Another primitive measure of surface orientation and depth is available from image flow (i.e., motion of an image on the retina of the eye). Image flow may be caused either by motion of objects in the world, or by motion of the eye through the world. The image flow of stationary objects caused by translation of the eye is inversely proportional to the distance from the eye to the point being observed. Thus, if eye rotation is zero, and the translational velocity of the eye is known, the focus of expansion is fixed, and image flow lines are defined by great circle arcs on the velocity egosphere that emanate from the focus of expansion and pass through the pixel in question [45]. Under these conditions, range to any stationary point in the world can be computed directly from image flow by the simple formula

$$R = \frac{v \sin A}{dA/dt} \tag{1}$$

where $R$ is the range to the point, $v$ is translational velocity vector of the eye, $A$ is the angle between the velocity vector

and the pixel covering the point. $dA/dt$ is the image flow rate at the pixel covering the point

When eye rotation is zero and $v$ is known, the flow rate $dA/dt$ can be computed locally for each pixel from temporal and spatial derivatives of image brightness along flow lines on the velocity egosphere. $dA/dt$ can also be computed from temporal crosscorrelation of brightness from adjacent pixels along flow lines.

When the eye fixates on a point, $dA/dt$ is equal to the rotation rate of the eye. Under this condition, the distance to the fixation point can be computed from (1), and the distance to other points may be computed from image flow relative to the fixation point.

If eye rotation is nonzero but known, the range to any stationary point in the world may be computed by a closed form formula of the form

$$R = F\left(x, y, T, W, \frac{\partial I}{\partial t}, \frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right) \tag{2}$$

where $x$ and $z$ are the image coordinates of a pixel, $T$ is the translational velocity vector of the camera in camera coordinates, $W$ is the rotational velocity vector of the camera in camera coordinates, and $I$ is the pixel brightness intensity. This type of function can be implemented locally and in parallel by a neural net for each image pixel [46].

Knowledge of eye velocity, both translational and rotational, may be computed by the vestibular system, the locomotion system, and/or high levels of the vision system. Knowledge of, rotational eye motion may either be used in the computation of range by (2), or can be used to transform sensor egosphere images into velocity egosphere coordinates where (1) applies. This can be accomplished mechanically by the vestibulo-ocular reflex, or electronically (or neuronally) by scrolling the input image through an angle determined by a function of data variables from the vestibular system and the ocular muscle stretch receptors. Virtual transformation of image coordinates can be accomplished using coordinate transform parameters located in each map pixel frame.

Depth from image flow enables creatures of nature, from fish and insects to birds and mammals, to maneuver rapidly through natural environments filled with complex obstacles without collision. Moving objects can be segmented from stationary by their failure to match world model predictions for stationary objects. Near objects can be segmented from distant by their differential flow rates.

Distance to surfaces may also be computed from stereo-vision. The angular disparity between images in two eyes separated by a known distance can be used to compute range. Depth from stereo is more complex than depth from image flow in that it requires identification of corresponding points in images from different eyes. Hence it cannot be computed locally. However, stereo is simpler than image flow in that it does not require eye translation and is not confounded by eye rotation or by moving objects in the world. The computation of distance from a combination of both motion and stereo is more robust, and hence psychophysically more vivid to the observer, than from either motion or stereo alone.

Distance to surfaces may also be computed from sonar or radar by measuring the time delay between emitting radiation and receiving an echo. Difficulties arise from poor angular resolution and from a variety of sensitivity, scattering, and multipath problems. Creatures such as bats and marine mammals use multispectral signals such as chirps and clicks to minimize confusion from these effects. Phased arrays and synthetic apertures may also be used to improve the resolution of radar or sonar systems.

All of the previous methods for perceiving surfaces are primitive in the sense that they compute depth directly from sensory input without recognizing entities or understanding anything about the scene. Depth measurements from primitive processes can immediately generate maps that can be used directly by the lower levels of the behavior generation hierarchy to avoid obstacles and approach surfaces.

Surface attributes such as position and orientation may also be computed from shading, shadows, and texture gradients. These methods typically depend on higher levels of visual perception such as geometric reasoning, recognition of objects, detection of events and states, and the understanding of scenes.

### *B. Recognition and Detection*

*Definition:* Recognition is the establishment of a one-to-one match, or correspondence, between a real world entity and a world model entity.

The process of recognition may proceed top-down, or bottom-up, or both simultaneously. For each entity in the world model, there exists a frame filled with information that can be used to predict attributes of corresponding entities observed in the world. The top-down process of recognition begins by hypothesizing a world model entity and comparing its predicted attributes with those of the observed entity. When the similarities and differences between predictions from the world model and observations from sensory processing are integrated over a space-time window that covers an entity, a matching, or crosscorrelation value is computed between the entity and the model. If the correlation value rises above a selected threshold, the entity is said to be recognized. If not, the hypothesized entity is rejected and another tried.

The bottom-up process of recognition consists of applying filters and masks to incoming sensory data, and computing image properties and attributes. These may then be stored in the world model, or compared with the properties and attributes of entities already in the world model. Both top-down and bottom-up processes proceed until a match is found, or the list of world model entities is exhausted. Many perceptual matching processes may operate in parallel at multiple hierarchical levels simultaneously.

If a SP module recognizes a specific entity, the WM at that level updates the attributes in the frame of that specific WM entity with information from the sensory system.

If the SP module fails to recognize a specific entity, but instead achieves a match between the sensory input and a generic world model entity, a new specific WM entity will be created with a frame that initially inherits the features of the generic entity. Slots in the specific entity frame can then be

updated with information from the sensory input.

If the SP module fails to recognize either a specific or a generic entity, the WM may create an "unidentified" entity with an empty frame. This may then be filled with information gathered from the sensory input.

When an unidentified entity occurs in the world model, the behavior generation system may (depending on other priorities) select a new goal to <identify the unidentified entity>. This may initiate an exploration task that positions and focuses the sensor systems on the unidentified entity, and possibly even probes and manipulates it, until a world model frame is constructed that adequately describes the entity. The sophistication and complexity of the exploration task depends on task knowledge about exploring things. Such knowledge may be very advanced and include sophisticated tools and procedures, or very primitive. Entities may, of course, simply remain labeled as "unidentified," or unexplained.

Event detection is analogous to entity recognition. Observed states of the real world are compared with states predicted by the world model. Similarities and differences are integrated over an event space-time window, and a matching, or cross-correlation value is computed between the observed event and the model event. When the crosscorrelation value rises above a given threshold, the event is detected.

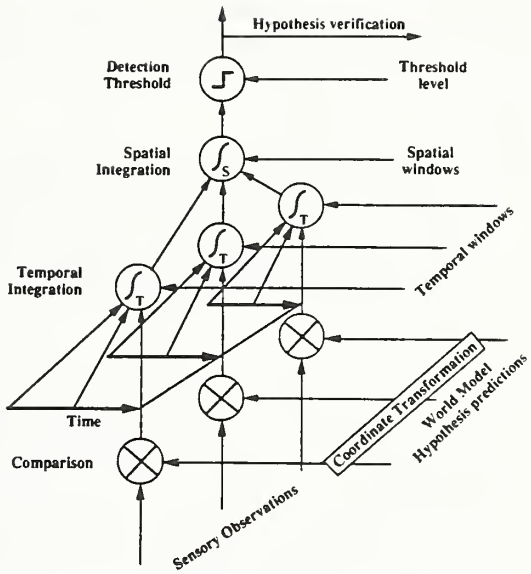



Fig. 15. Each sensory processing SP module consists of the following. 1) A set of comparators that compare sensory observations with world model predictions, 2) a set of temporal integrators that integrate similarities and differences, 3) a set of spatial integrators that fuse information from different sensory data streams, and 4) a set of threshold detectors that recognize entities and detect events.

### C. The Context of Perception

If, as suggested in Fig. 5, there exists in the world model at every hierarchical level a short term memory in which is stored a temporal history consisting of a series of past values of time dependent entity and event attributes and states, it can be assumed that at any point in time, an intelligent system has a record in its short term memory of how it reached its current state. Figs. 5 and 6 also imply that, for every planner in each behavior generating BG module at each level, there exists a plan, and that each executor is currently executing the first step in its respective plan. Finally, it can be assumed that the knowledge in all these plans and temporal histories, and all the task, entity, and event frames referenced by them, is available in the world model.

Thus it can be assumed that an intelligent system almost always knows where it is on a world map, knows how it got there, where it is going, what it is doing, and has a current list of entities of attention, each of which has a frame of attributes (or state variables) that describe the recent past, and provide a basis for predicting future states. This includes a prediction of what objects will be visible, where and how object surfaces will appear, and which surface boundaries, vertices, and points will be observed in the image produced by the sensor system. It also means that the position and motion of the eyes, ears, and tactile sensors relative to surfaces and objects in the world are known, and this knowledge is available to be used by the sensory processing system for constructing maps and overlays, recognizing entities, and detecting events.

Were the aforementioned not the case, the intelligent system would exist in a situation analogous to a person who suddenly awakens at an unknown point in space and time. In such cases, it typically is necessary even for humans to perform a series of tasks designed to "regain their bearings", i.e., to bring their world model into correspondence with the state of the external world, and to initialize plans, entity frames, and system state variables.

It is, of course, possible for an intelligent creature to function in a totally unknown environment, but not well, and not for long. Not well, because every intelligent creature makes much good use of the historical information that forms the context of its current task. Without information about where it is, and what is going on, even the most intelligent creature is severely handicapped. Not for long, because the sensory processing system continuously updates the world model with new information about the current situation and its recent historical development, so that, within a few seconds, a functionally adequate map and a usable set of entity state variables can usually be acquired from the immediately surrounding environment.

### D. Sensory Processing SP Modules

At each level of the proposed architecture, there are a number of computational nodes. Each of these contains an SP module, and each SP module consists of four sublevels, as shown in Fig. 15.

*Sublevel 1—Comparison:* Each comparison submodule matches an observed sensory variable with a world model prediction of that variable. This comparison typically involves an arithmetic operation, such as multiplication or subtraction, which yields a measure of similarity and difference between an observed variable and a predicted variable. Similarities indicate the degree to which the WM predictions are correct, and hence are a measure of the correspondence between the world model and reality. Differences indicate a lack of

correspondence between world model predictions and sensory observations. Differences imply that either the sensor data or world model is incorrect. Difference images from the comparator go three places:

1) They are returned directly to the WM for real-time local pixel attribute updates. This produces a tight feedback loop whereby the world model predicted image becomes an array of Kalman filter state-estimators. Difference images are thus error signals by which each pixel of the predicted image can be trained to correspond to current sensory input.
2) They are also transmitted upward to the integration sublevels where they are integrated over time and space in order to recognize and detect global entity attributes. This integration constitutes a summation, or chunking, of sensory data into entities. At each level, lower order entities are "chunked" into higher order entities, i.e., points are chunked into lines, lines into surfaces, surfaces into objects, objects into groups, etc.
3) They are transmitted to the VJ module at the same level where statistical parameters are computed in order to assign confidence and believability factors to pixel entity attribute estimates.

*Sublevel 2—Temporal integration:* Temporal integration submodules integrate similarities and differences between predictions and observations over intervals of time. Temporal integration submodules operating just on sensory data can produce a summary, such as a total, or average, of sensory information over a given time window. Temporal integrator submodules operating on the similarity and difference values computed by comparison submodules may produce temporal crosscorrelation and covariance functions between the model and the observed data. These correlation and covariance functions are measures of how well the dynamic properties of the world model entity match those of the real world entity. The boundaries of the temporal integration window may be derived from world model prediction of event durations, or form behavior generation parameters such as sensor fixation periods.

*Sublevel 3—Spatial integration:* Spatial integrator submodules integrate similarities and differences between predictions and observations over regions of space. This produces spatial crosscorrelation or convolution functions between the model and the observed data. Spatial integration summarizes sensory information from multiple sources at a single point in time. It determines whether the geometric properties of a world model entity match those of a real world entity. For example, the product of an edge operator and an input image may be integrated over the area of the operator to obtain the correlation between the image and the edge operator at a point. The limits of the spatial integration window may be determined by world model predictions of entity size. In some cases, the order of temporal and spatial integration may be reversed, or interleaved.

*Sublevel 4—Recognition/Detection threshold:* When the spatiotemporal correlation function exceeds some threshold, object recognition (or event detection) occurs. For example, if the spatiotemporal summation over the area of an edge operator exceeds threshold, an edge is said to be detected at the center of the area.

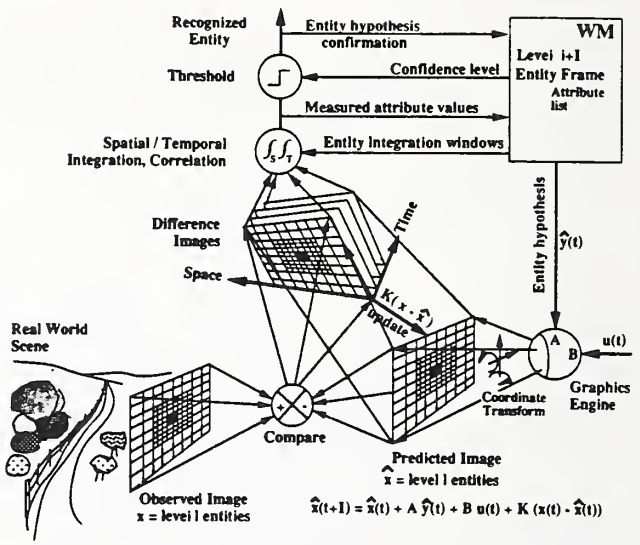



Fig. 16. Interaction between world model and sensory processing. Difference images are generator by comparing predicted images with observed images. Feedback of differences produces a Kalman best estimate for each data variable in the world model. Spatial and temporal integration produce crosscorrelation functions between the estimated attributes in the world model and the real-world attributes measured in the observed image. When the correlation exceeds threshold, entity recognition occurs.

Fig. 16 illustrates the nature of the SP-WM interactions between an intelligent vision system and the world model at one level. On the left of Fig. 16, the world of reality is viewed through the window of an egosphere such as exists in the primary visual cortex. On the right is a world model consisting of: 1) a symbolic entity frame in which entity attributes are stored, and 2) an iconic predicted image that is registered in real-time with the observed sensory image. In the center of Fig. 16, is a comparator where the expected image is subtracted from (or otherwise compared with) the observed image.

The level($i$) predicted image is initialized by the equivalent of a graphics engine operating on symbolic data from frames of entities hypothesized at level($i+1$). The predicted image is updated by differences between itself and the observed sensory input. By this process, the predicted image becomes the world model's "best estimate prediction" of the incoming sensory image, and a high speed loop is closed between the WM and SP modules at level($i$).

When recognition occurs in level ($i$), the world model level($i+1$) hypothesis is confirmed and both level($i$) and level($i+1$) symbolic parameters that produced the match are updated in the symbolic database. This closes a slower, more global, loop between WM and SP modules through the symbolic entity frames of the world model. Many examples of this type of looping interaction can be found in the model matching and model-based recognition literature [47]. Similar closed loop filtering concepts have been used for years for signal detection, and for dynamic systems modeling in aircraft flight control systems. Recently they have been applied to

high speed visually guided driving of an autonomous ground vehicle [48].

The behavioral performance of intelligent biological creatures suggests that mechanisms similar to those shown in Figs. 15 and 16 exist in the brain. In biological or neural network implementations, SP modules may contain thousands, even millions, of comparison submodules, temporal and spatial integrators, and threshold submodules. The neuroanatomy of the mammalian visual system suggests how maps with many different overlays, as well as lists of symbolic attributes, could be processed in parallel in real-time. In such structures it is possible for multiple world model hypotheses to be compared with sensory observations at multiple hierarchical levels, all simultaneously.

### E. World Model Update

Attributes in the world model predicted image may be updated by a formula of the form

$$\hat{\boldsymbol{x}}(t+1) = \hat{\boldsymbol{x}}(t) + \boldsymbol{A}\hat{\boldsymbol{y}}(t) + \boldsymbol{B}\boldsymbol{u}(t) + \boldsymbol{K}(t)[\boldsymbol{x}(t) - \hat{\boldsymbol{x}}(t)] \tag{3}$$

where $\hat{\boldsymbol{x}}(t)$ is the best estimate vector of world model $i$-order entity attributes at time $t$, $A$ is a matrix that computes the expected rate of change of $\hat{\boldsymbol{x}}(t)$ given the current best estimate of the $i+1$ order entity attribute vector $\hat{\boldsymbol{y}}(t)$, $B$ is a matrix that computes the expected rate of change of $\hat{\boldsymbol{x}}(t)$ due to external input $u(t)$, and $K(t)$ is a confidence factor vector for updating $\hat{\boldsymbol{x}}(t)$. The value of $K(t)$ may be computed by a formula of the form

$$\boldsymbol{K}(t) = K_s(j,t)[1 - K_m(j,t)] \tag{4}$$

where $K_s(j,t)$ is the confidence in the sensory observation of the $j$th real world attribute $x(j,t)$ at time $t$, $0 \leq K_s(j,t) \leq 1$ $K_m(j,t)$ is the confidence in the world model prediction of the $j$th attribute at time $t$ $0 \leq K_m(j,t) \leq 1$.

The confidence factors ($K_m$ and $K_s$) in formula (4) may depend on the statistics of the correspondence between the world model entity and the real world entity (e.g. the number of data samples, the mean and variance of $[\boldsymbol{x}(t) - \hat{\boldsymbol{x}}(t)]$, etc.). A high degree of correlation between $x(t)$ and $[\hat{\boldsymbol{x}}(t)]$ in both temporal and spatial domains indicates that entities or events have been correctly recognized, and states and attributes of entities and events in the world model correspond to those in the real world environment. World model data elements that match observed sensory data elements are reinforced by increasing the confidence, or believability factor, $K_m(j,t)$ for the entity or state at location $j$ in the world model attribute lists. World model entities and states that fail to match sensory observations have their confidence factors $K_m(j,t)$ reduced. The confidence factor $K_s(j,t)$ may be derived from the signal-to-noise ratio of the $j$th sensory data stream.

The numerical value of the confidence factors may be computed by a variety of statistical methods such Baysian or Dempster–Shafer statistics.

### F. The Mechanisms of Attention

*Theorem:* Sensory processing is an active process that is directed by goals and priorities generated in the behavior generating system.

In each node of the intelligent system hierarchy, the behavior generating BG modules request information needed for the current task from sensory processing SP modules. By means of such requests, the BG modules control the processing of sensory information and focus the attention of the WM and SP modules on the entities and regions of space that are important to success in achieving behavioral goals. Requests by BG modules for specific types of information cause SP modules to select particular sensory processing masks and filters to apply to the incoming sensory data. Requests from BG modules enable the WM to select which world model data to use for predictions, and which prediction algorithm to apply to the world model data. BG requests also define which correlation and differencing operators to use, and which spatial and temporal integration windows and detection thresholds to apply.

Behavior generating BG modules in the attention subsystem also actively point the eyes and ears, and direct the tactile sensors of antennae, fingers, tongue, lips, and teeth toward objects of attention. BG modules in the vision subsystem control the motion of the eyes, adjust the iris and focus, and actively point the fovea to probe the environment for the visual information needed to pursue behavioral goals [49], [50]. Similarly, BG modules in the auditory subsystem actively direct the ears and tune audio filters to mask background noises and discriminate in favor of the acoustic signals of importance to behavioral goals.

Because of the active nature of the attention subsystem, sensor resolution and sensitivity is not uniformly distributed, but highly focused. For example, receptive fields of optic nerve fibers from the eye are several thousand times more densely packed in the fovea than near the periphery of the visual field. Receptive fields of touch sensors are also several thousand times more densely packed in the finger tips and on the lips and tongue, than on other parts of the body such as the torso.

The active control of sensors with nonuniform resolution has profound impact on the communication bandwidth, computing power, and memory capacity required by the sensory processing system. For example, there are roughly 500 000 fibers in the the optic nerve from a single human eye. These fibers are distributed such that about 100 000 are concentrated in the ±1.0 degree foveal region with resolution of about 0.007 degrees. About 100 000 cover the surrounding ±3 degree region with resolution of about 0.02 degrees. 100 000 more cover the surrounding ±10 degree region with resolution of 0.07 degrees. 100 000 more cover the surrounding 30 degree region with a resolution of about 0.2 degrees. 100 000 more cover the remaining ±80 degree region with resolution of about 0.7 degree [51]. The total number of pixels is thus about 500 000 pixels, or somewhat less than that contained in two standard commercial TV images. Without nonuniform resolution, covering the entire visual field with the resolution of the fovea would require the number of pixels in about 6 000

standard TV images. Thus, for a vision sensory processing system with any given computing capacity, active control and nonuniform resolution in the retina can produce more than three orders of magnitude improvement in image processing capability.

SP modules in the attention subsystem process data from low-resolution wide-angle sensors to detect regions of interest, such as entities that move, or regions that have discontinuities (edges and lines), or have high curvature (corners and intersections). The attention BG modules then actively maneuver the eyes, fingers, and mouth so as to bring the high resolution portions of the sensory systems to bear precisely on these points of attention. The result gives the subjective effect of high resolution everywhere in the sensory field. For example, wherever the eye looks, it sees with high resolution, for the fovea is always centered on the item of current interest.

The act of perception involves both sequential and parallel operations. For example, the fovea of the eye is typically scanned sequentially over points of attention in the visual field [52]. Touch sensors in the fingers are actively scanned over surfaces of objects, and the ears may be pointed toward sources of sound. While this sequential scanning is going on, parallel recognition processes hypothesize and compare entities at all levels simultaneously.

### G. The Sensory Processing Hierarchy

It has long been recognized that sensory processing occurs in a hierarchy of processing modules, and that perception proceeds by "chunking", i.e., by recognizing patterns, groups, strings, or clusters of points at one level as a single feature, or point in a higher level, more abstract space. It also has been observed that this chunking process proceeds by about an order of magnitude per level, both spatially and temporally [17], [18]. Thus, at each level in the proposed architecture, SP modules integrate, or chunk, information over space and time by about an order of magnitude.

Fig. 17 describes the nature of the interactions hypothesized to take place between the sensory processing and world modeling modules at the first four levels, as the recognition process proceeds. The functional properties of the SP modules are coupled to, and determined by, the predictions of the WM modules in their respective processing nodes. The WM predictions are, in turn, effected by states of the BG modules.

*Hypothesis:* There exist both iconic (maps) and symbolic (entity frames) at all levels of the SP/WM hierarchy of the mammalian vision system.

Fig. 18 illustrates the concept stated in this hypothesis. Visual input to the retina consists of photometric brightness and color intensities measured by rods and cones. Brightness intensities are denoted by $I(k, AZ, EL, t)$, where $I$ is the brightness intensity measured at time $t$ by the pixel at sensor egosphere azimuth $AZ$ and elevation $EL$ of eye (or camera) $k$. Retinal intensity signals $I$ may vary over time intervals on the order of a millisecond or less.

Image preprocessing is performed on the retina by horizontal, bipolar, amacrine, and ganglion cells. Center-surround receptive fields ("on-center" and "off-center") detect both spatial and temporal derivatives at each point in the visual

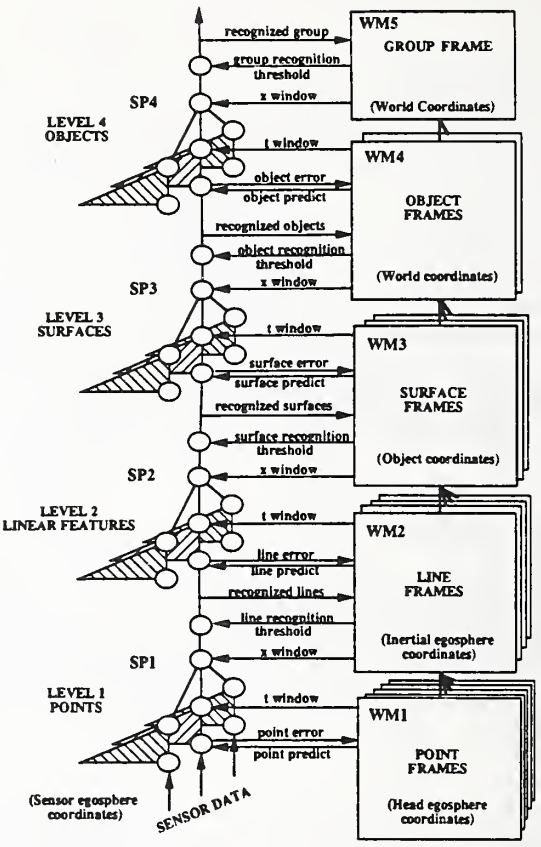



Fig. 17. The nature of the interactions that take place between the world model and sensory processing modules. At each level, predicted entities are compared with bo observed. Differences are returned as errors directly to the world model to update the model. Correlations are forwarded upward to be integrated over time and space windows provided by the world model. Correlations that exceed threshold are d recognized as entities.

field. Outputs from the retina carried by ganglion cell axons become input to sensory processing level 1 as shown in Fig. 18. Level 1 inputs correspond to events of a few milliseconds duration.

It is hypothesized that in the mammalian brain, the level 1 vision processing module consists of the neurons in the lateral geniculate bodies, the superior colliculus, and the primary visual cortex ($V1$). Optic nerve inputs from the two eyes are overlaid such that the visual fields from left and right eyes are in registration. Data from stretch sensors in the ocular muscles provides information to the superior colliculus about eye convergence, and pan, tilt, and roll of the retina relative to the head. This allows image map points in retinal coordinates to be transformed into image map points in head coordinates (or vice versa) so that visual and acoustic position data can be registered and fused [41], [42]. In $V1$, registration of corresponding pixels from two separate eyes on single neurons also provides the basis for range from stereo to be computed for each pixel [31].

At level 1, observed point entities are compared with predicted point entities. Similarities and differences are integrated into linear entities. Strings of level 1 input events are integrated into level 1 output events spanning a few tens of milliseconds. Level 1 outputs become level 2 inputs.

The level 2 vision processing module is hypothesized to

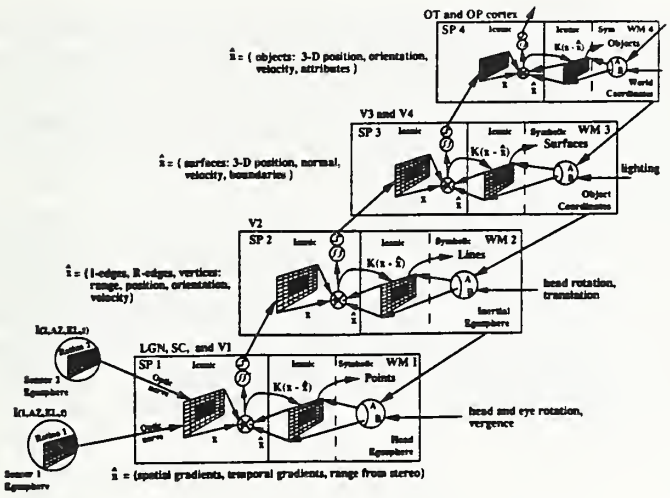



Fig. 18. Hypothesized correspondence between levels in the proposed model and neuranatomical structures in the mammalian vision system. At each level, the WM module contains both iconic and symbolic representations. At each level, the SP module compares the observed image with a predicted image. At each level, both iconic and symbolic world models are updated, and map overlays are computed. LGN is the lateral geniculate nuclei, OT is the occipital–temporal, OP is the occipital–parietal, and SC is the superior colliculus.

consist of neurons in the secondary visual cortex (V2). At level 2, observed linear entities are compared with predicted linear entities. Similarities and differences are integrated into surface entities. Some individual neurons indicate edges and lines at particular orientations. Other neurons indicate edge points, curves, trajectories, vertices, and boundaries.

Input to the world model from the vestibular system indicates the direction of gravity and the rotation of the head. This allows the level 2 world model to transform head egosphere representations into inertial egosphere coordinates where the world is perceived to be stationary despite rotation of the sensors.

Acceleration data from the vestibular system, combined with velocity data from the locomotion system, provide the basis for estimating both rotary and linear eye velocity, and hence image flow direction. This allows the level 2 world model to transform head egosphere representations into velocity egosphere coordinates where depth from image flow can be computed. Center-surround receptive fields along image flow lines can be subtracted from each other to derive spatial derivatives in the flow direction. At each point where the spatial derivative in the flow direction is nonzero, spatial and temporal derivatives can be combined with knowledge of eye velocity to compute the image flow rate $dA/dt$ [45]. Range to each pixel can then be computed directly, and in parallel, from local image data using formula (1) or (2).

The previous egosphere transformations do not necessarily imply that neurons are physically arranged in inertial or velocity egosphere coordinates on the visual cortex. If that were true, it would require that the retinal image be scrolled over the cortex, and there is little evidence for this, at least in $V1$ and $V2$. Instead, it is conjectured that the neurons that make up both observed and predicted iconic images exist on the visual cortex in retinotopic, or sensor egosphere, coordinates. The velocity and inertial egosphere coordinates for each pixel are defined by parameters in the symbolic entity frame of each pixel. The inertial, velocity (and perhaps head) egospheres may thus be "virtual" egospheres. The position of any pixel on any egosphere can be computed by using the transformation parameters in the map pixel frame as an indirect address offset. This allows velocity and inertial egosphere computations to be performed on neural patterns that are physically represented in sensor egosphere coordinates.

The possibility of image scrolling cannot be ruled out, however, particularly at higher levels. It has been observed that both spatial and temporal retinotopic specificity decreases about two orders of magnitude from $V1$ to $V4$ [54]. This is consistent with scrolling.

Strings of level 2 input events are integrated into level 3 input events spanning a few hundreds of milliseconds.

The level 3 vision processing module is hypothesized to reside in areas $V3$ and $V4$ of the visual cortex. Observed surface entities are compared with predicted surface entities. Similarities and differences are integrated to recognize object entities. Cells that detect texture and motion of regions in specific directions provide indication of surface boundaries and depth discontinuities. Correlations and differences between world model predictions and sensory observations of surfaces give rise to meaningful image segmentation and recognition of surfaces. World model knowledge of lighting and texture allow computation of surface orientation, discontinuities, boundaries, and physical properties.

Strings of level 3 input events are integrated into level 4 input events spanning a few seconds. (This does not necessarily imply that it takes seconds to recognize surfaces, but that both patterns of motion that occupy a few seconds, and surfaces, are recognized at level 3. For example, the recognition of a gesture, or dance step, might occur at this level.)

World model knowledge of the position of the self relative to surfaces enables level 3 to compute offset variables for each pixel that transform it from inertial egosphere coordinates into object coordinates.

The level 4 vision processing module is hypothesized to reside in the posterior inferior temporal and ventral intraparietal regions of visual cortex. At level 4, observed objects are compared with predicted objects. Correlations and differences between world model predictions and sensory observations of objects allows shape, size, and orientation, as well as location, velocity, rotation, and size-changes of objects to be recognized and measured.

World model input from the locomotion and navigation systems allow level 4 to transform object coordinates into world coordinates. Strings of level 4 input events are grouped into level 5 input events spanning a few tens of seconds.

Level 5 vision is hypothesized to reside in the visual association areas of the parietal and temporal cortex. At level 5, observed groups of objects are compared with predicted

groups. Correlations are integrated into group$^2$ entities. Strings of level 5 input events are detected as level 5 output events spanning a few minutes. For example, in the anterior inferior temporal region particular groupings of objects such as eyes, nose, and mouth are recognized as faces. Groups of fingers can be recognized as hands, etc. In the parietal association areas, map positions, orientations, rotations of groups of objects are detected. At level 5, the world model map has larger span and lower resolution than level 4.

At level 6, clusters of group$^2$ entities are recognized as group$^3$ entities, and strings of level 6 input events are grouped into level 6 output events spanning a few tens of minutes. The world model map at level 7 has larger span and lower resolution than at level 6.

At level 7, strings of level 7 input events are grouped into level 7 output events spanning a few hours.

It must be noted that the neuroanatomy of the mammalian vision system is much more convoluted than suggested by Fig. 18. Van Essen [53] has compiled a list of 84 identified or suspected pathways connecting 19 visual areas. Visual processing is accomplished in at least two separate subsystems that are not differentiated in Fig. 18. The subsystem that includes the temporal cortex emphasizes the recognition of entities and their attributes such as shape, color, orientation, and grouping of features. The subsystem that includes the parietal cortex emphasizes spatial and temporal relationships such as map positions, timing of events, velocity, and direction of motion [54]. It should also be noted that analogous figures could be drawn for other sensory modalities such as hearing and touch.

### H. Gestalt Effects

When an observed entity is recognized at a particular hierarchical level, its entry into the world model provides predictive support to the level below. The recognition output also flows upward where it narrows the search at the level above. For example, a linear feature recognized and entered into the world model at level 2, can be used to generate expected points at level 1. It can also be used to prune the search tree at level 3 to entities that contain that particular type of linear feature. Similarly, surface features at level 3 can generate specific expected linear features at level 2, and limit the search at level 4 to objects that contain such surfaces, etc. The recognition of an entity at any level thus provides to both lower and higher levels information that is useful in selecting processing algorithms and setting spatial and temporal integration windows to integrate lower level features into higher level chunks.

If the correlation function at any level falls below threshold, the current world model entity or event at that level will be rejected, and others tried. When an entity or event is rejected, the rejection also propagates both upward and downward, broadening the search space at both higher and lower levels.

At each level, the SP and WM modules are coupled so as to form a feedback loop that has the properties of a relaxation process, or phase-lock loop. WM predictions are compared with SP observations, and the correlations and differences are fed back to modify subsequent WM predictions. WM predictions can thus be "servoed" into correspondence with the SP observations. Such looping interactions will either converge to a tight correspondence between predictions and observations, or will diverge to produce a definitive set of irreconcilable differences.

Perception is complete only when the correlation functions at all levels exceed threshold simultaneously. It is the nature of closed loop processes for lock-on to occur with a positive "snap". This is especially pronounced in systems with many coupled loops that lock on in quick succession. The result is a gestalt "aha" effect that is characteristic of many human perceptions.

### I. Flywheeling, Hysteresis, and Illusion

Once recognition occurs, the looping process between SP and WM acts as a tracking filter. This enables world model predictions to track real world entities through noise, data dropouts, and occlusions.

In the system described previously, recognition will occur when the first hypothesized entity exceeds threshold. Once recognition occurs, the search process is suppressed, and the thresholds for all competing recognition hypotheses are effectively raised. This creates a hysteresis effect that tends to keep the WM predictions locked onto sensory input during the tracking mode. It may also produce undesirable side effects, such as a tendency to perceive only what is expected, and a tendency to ignore what does not fit preconceived models of the world.

In cases where sensory data is ambiguous, there is more than one model that can match a particular observed object. The first model that matches will be recognized, and other models will be suppressed. This explains the effects produced by ambiguous figures such as the Necker cube.

Once an entity has been recognized, the world model projects its predicted appearance so that it can be compared with the sensory input. If this predicted information is added to, or substituted for, sensory input, perception at higher levels will be based on a mix of sensory observations and world model predictions. By this mechanism, the world model may fill in sensory data that is missing, and provide information that may be left out of the sensory data. For example, it is well known that the audio system routinely "flywheels" through interruptions in speech data, and fills-in over noise bursts.

This merging of world model predictions with sensory observations may account for many familiar optical illusions such as subjective contours and the Ponzo illusion. In pathological cases, it may also account for visions and voices, and an inability to distinguish between reality and imagination. Merging of world model prediction with sensory observation is what Grossberg calls "adaptive resonance" [55].

## XV. Value Judgments

Value judgments provide the criteria for making intelligent

choices. Value judgments evaluate the costs, risks, and benefits of plans and actions, and the desirability, attractiveness, and uncertainty of objects and events. Value judgment modules produce evaluations that can be represented as value state-variables. These can be assigned to the attribute lists in entity frames of objects, persons, events, situations, and regions of space. They can also be assigned to the attribute lists of plans and actions in task frames. Value state-variables can label entities, tasks, and plans as good or bad, costly or inexpensive, as important or trivial, as attractive or repulsive, as reliable or uncertain. Value state-variables can also be used by the behavior generation modules both for planning and executing actions. They provide the criteria for decisions about which coarse of action to take [56].

*Definition:* Emotions are biological value state-variables that provide estimates of good and bad.

Emotion value state-variables can be assigned to the attribute lists of entities, events, tasks, and regions of space so as to label these as good or bad, as attractive or repulsive, etc. Emotion value state-variables provide criteria for making decisions about how to behave in a variety of situations. For example, objects or regions labeled with fear can be avoided, objects labeled with love can be pursued and protected, those labeled with hate can be attacked, etc. Emotional value judgments can also label tasks as costly or inexpensive, risky or safe.

*Definition:* Priorities are value state-variables that provide estimates of importance.

Priorities can be assigned to task frames so that BG planners and executors can decide what to do first, how much effort to spend, how much risk is prudent, and how much cost is acceptable, for each task.

*Definition:* Drives are value state-variables that provide estimates of need.

Drives can be assigned to the self frame, to indicate internal system needs and requirements. In biological systems, drives indicate levels of hunger, thirst, and sexual arousal. In mechanical systems, drives might indicate how much fuel is left, how much pressure is in a boiler, how many expendables have been consumed, or how much battery charge is remaining.

### A. *The Limbic System*

In animal brains, value judgment functions are computed by the limbic system. Value state-variables produced by the limbic system include emotions, drives, and priorities. In animals and humans, electrical or chemical stimulation of specific limbic regions (i.e., value judgment modules) has been shown to produce pleasure and pain as well as more complex emotional feelings such as fear, anger, joy, contentment, and despair. Fear is computed in the posterior hypothalamus. Anger and rage are computed in the amygdala. The insula computes feelings of contentment, and the septal regions produce joy and elation. The perifornical nucleus of the hypothalamus computes punishing pain, the septum pleasure, and the pituitary computes the body's priority level of arousal in response to danger and stress [57].

The drives of hunger and thirst are computed in the limbic system's medial and lateral hypothalamus. The level of sexual arousal is computed by the anterior hypothalamus. The control of body rhythms, such as sleep-awake cycles, are computed by the pineal gland. The hippocampus produces signals that indicate what is important and should be remembered, or what is unimportant and can safely be forgotten. Signals from the hippocampus consolidate (i.e., make permanent) the storage of sensory experiences in long term memory. Destruction of the hippocampus prevents memory consolidation [58].

In lower animals, the limbic system is dominated by the sense of smell and taste. Odor and taste provides a very simple and straight forward evaluation of many objects. For example, depending on how something smells, one should either eat it, fight it, mate with it, or ignore it. In higher animals, the limbic system has evolved to become the seat of much more sophisticated value judgments, including human emotions and appetites. Yet even in humans, the limbic system retains its primitive function of evaluating odor and taste, and there remains a close connection between the sense of smell and emotional feelings.

Input and output fiber systems connect the limbic system to sources of highly processed sensory data as well as to high level goal selection centers. Connections with the frontal cortex suggests that the value judgment modules are intimately involved with long range planning and geometrical reasoning. Connections with the thalamus suggests that the limbic value judgment modules have access to high level perceptions about objects, events, relationships, and situations; for example, the recognition of success in goal achievement, the perception of praise or hostility, or the recognition of gestures of dominance or submission. Connections with the reticular formation suggests that the limbic VJ modules are also involved in computing confidence factors derived from the degree of correlation between predicted and observed sensory input. A high degree of correlation produces emotional feelings of confidence. Low correlation between predictions and observations generates feelings of fear and uncertainty.

The limbic system is an integral and substantial part of the brain. In humans, the limbic system consists of about 53 emotion, priority, and drive submodules linked together by 35 major nerve bundles [57].

### B. *Value State-Variables*

It has long been recognized by psychologists that emotions play a central role in behavior. Fear leads to flight, hate to rage and attack. Joy produces smiles and dancing. Despair produces withdrawal and despondent demeanor. All creatures tend to repeat what makes them feel good, and avoid what they dislike. All attempt to prolong, intensify, or repeat those activities that give pleasure or make the self feel confident, joyful, or happy. All try to terminate, diminish, or avoid those activities that cause pain, or arouse fear, or revulsion.

It is common experience that emotions provide an evaluation of the state of the world as perceived by the sensory system. Emotions tell us what is good or bad, what is attractive or repulsive, what is beautiful or ugly, what is loved or hated, what provokes laughter or anger, what smells sweet or rotten,

what feels pleasurable, and what hurts.

It is also widely known that emotions affect memory. Emotionally traumatic experiences are remembered in vivid detail for years, while emotionally nonstimulating everyday sights and sounds are forgotten within minutes after they are experienced.

Emotions are popularly believed to be something apart from intelligence—irrational, beyond reason or mathematical analysis. The theory presented here maintains the opposite. In this model, emotion is a critical component of biological intelligence, necessary for evaluating sensory input, selecting goals, directing behavior, and controlling learning.

It is widely believed that machines cannot experience emotion, or that it would be dangerous, or even morally wrong to attempt to endow machines with emotions. However, unless machines have the capacity to make value judgments (i.e., to evaluate costs, risks, and benefits, to decide which course of action, and what expected results, are good, and which are bad) machines can never be intelligent or autonomous. What is the basis for deciding to do one thing and not another, even to turn right rather than left, if there is no mechanism for making value judgments? Without value judgments to support decision making, nothing can be intelligent, be it biological or artificial.

Some examples of value state-variables are listed below, along with suggestions of how they might be computed. This list is by no means complete.

*Good* is a high level positive value state-variable. It may be assigned to the entity frame of any event, object, or person. It can be computed as a weighted sum, or spatiotemporal integration, of all other positive value state-variables assigned to the same entity frame.

*Bad* is a high level negative value state-variable. It can be computed as a weighted sum, or spatiotemporal integration, of all other negative value state-variables assigned to an entity frame.

*Pleasure:* Physical pleasure is a mid-level internal positive value state-variable that can be assigned to objects, events, or specific regions of the body. In the latter case, pleasure may be computed indirectly as a function of neuronal sensory inputs from specific regions of the body. Emotional pleasure is a high level internal positive value state-variable that can be computed as a function of highly processed information about situations in the world.

*Pain:* Physical pain is a low level internal negative value state-variable that can be assigned to specific regions of the body. It may be computed directly as a function of inputs from pain sensors in specific regions of the body. Emotional pain is a high level internal negative value state-variable that may be computed indirectly from highly processed information about situations in the world.

*Success_observed* is a positive value state-variable that represents the degree to which task goals are met, plus the amount of benefit derived therefrom.

*Success_expected* is a value state-variable that indicates the degree of expected success (or the estimated probability of success). It may be stored in a task frame, or computed during planning on the basis of world model predictions. When compared with success_observed it provides a base-line for measuring whether goals were met on, behind, or ahead of schedule; at, over, or under estimated costs; and with resulting benefits equal to, less than, or greater than those expected.

*Hope* is a positive value state-variable produced when the world model predicts a future success in achieving a good situation or event. When high hope is assigned to a task frame, the BG module may intensify behavior directed toward completing the task and achieving the anticipated good situation or event.

*Frustration* is a negative value state-variable that indicates an inability to achieve a goal. It may cause a BG module to abandon an ongoing task, and switch to an alternate behavior. The level of frustration may depend on the priority attached to the goal, and on the length of time spent in trying to achieve it.

*Love* is a positive value state-variable produced as a function of the perceived attractiveness and desirability of an object or person. When assigned to the frame of an object or person, it tends to produce behavior designed to approach, protect, or possess the loved object or person.

*Hate* is a negative value state-variable produced as a function of pain, anger, or humiliation. When assigned to the frame of an object or person, hate tends to produce behavior designed to attack, harm, or destroy the hated object or person.

*Comfort* is a positive value state-variable produced by the absence of (or relief from) stress, pain, or fear. Comfort can be assigned to the frame of an object, person, or region of space that is safe, sheltering, or protective. When under stress or in pain, an intelligent system may seek out places or persons with entity frames that contain a large comfort value.

*Fear* is a negative value state-variable produced when the sensory processing system recognizes, or the world model predicts, a bad or dangerous situation or event. Fear may be assigned to the attribute list of an entity, such as an object, person, situation, event, or region of space. Fear tends to produce behavior designed to avoid the feared situation, event, or region, or flee from the feared object or person.

*Joy* is a positive value state-variable produced by the recognition of an unexpectedly good situation or event. It is assigned to the self-object frame.

*Despair* is a negative value state-variable produced by world model predictions of unavoidable, or unending, bad situations or events. Despair may be caused by the inability of the behavior generation planners to discover an acceptable plan for avoiding bad situations or events.

*Happiness* is a positive value state-variable produced by sensory processing observations and world model predictions of good situations and events. Happiness can be computed as a function of a number of positive (rewarding) and negative (punishing) value state-variables.

*Confidence* is an estimate of probability of correctness. A confidence state-variable may be assigned to the frame of any entity in the world model. It may also be assigned to the self frame, to indicate the level of confidence that a creature has in its own capabilities to deal with a situation. A high value of confidence may cause the BG hierarchy to behave confidently or aggressively.

*Uncertainty* is a lack of confidence. Uncertainty assigned to the frame of an external object may cause attention to be

directed toward that object in order to gather more information about it. Uncertainty assigned to the self-object frame may cause the behavior generating hierarchy to be timid or tentative.

It is possible to assign a real nonnegative numerical scalar value to each value state-variable. This defines the degree, or amount, of that value state-variable. For example, a positive real value assigned to "good" defines how good; i.e., if

$$e := \text{"good"} \text{ and } 0 \leq e \leq 10 \tag{5}$$

then, $e = 10$ is the "best" evaluation possible.

Some value state-variables can be grouped as conjugate pairs. For example, good-bad, pleasure-pain, success-fail, love-hate, etc. For conjugate pairs, a positive real value means the amount of the good value, and a negative real value means the amount of the bad value.

For example, if

$$e := \text{"good-bad"} \text{ and } -10 \leq e \leq +10$$

then $e = 5$ is good $e = 6$ is better $e = 10$ is best $e = -4$ is bad $e = -7$ is worse $e = -10$ is worst $e = 0$ is neither good nor bad.

Similarly, in the case of pleasure-pain, the larger the positive value, the better it feels. The larger the negative value, the worse it hurts. For example, if

$$e := \text{"pleasure-pain"}$$

then $e = 5$ is pleasurable $e = 10$ is ecstasy $e = -5$ is painful $e = -10$ is agony $e = 0$ is neither pleasurable nor painful.

The positive and negative elements of the conjugate pair may be computed separately, and then combined.

### C. VJ Modules

Value state-variables are computed by value judgment functions residing in VJ modules. Inputs to VJ modules describe entities, events, situations, and states. VJ value judgment functions compute measures of cost, risk, and benefit. VJ outputs are value state-variables.

*Theorem:* The VJ value judgment mechanism can be defined as a mathematical or logical function of the form

$$\boldsymbol{E} = \boldsymbol{V}(\boldsymbol{S}) \tag{6}$$

where $\boldsymbol{E}$ is an output vector of value state-variables, $\boldsymbol{V}$ is a value judgment function that computes $\boldsymbol{E}$ given $\boldsymbol{S}$, $\boldsymbol{S}$ is an input state vector defining conditions in the world model, including the self. The components of $\boldsymbol{S}$ are entity attributes describing states of tasks, objects, events, or regions of space. These may be derived either from processed sensory information, or from the world model.

The value judgment function $\boldsymbol{V}$ in the VJ module computes a numerical scalar value (i.e., an evaluation) for each component of $\boldsymbol{E}$ as a function of the input state vector $\boldsymbol{S}$, $\boldsymbol{E}$ is a time dependent vector. The components of $\boldsymbol{E}$ may be assigned to attributes in the world model frame of various entities, events, or states.

If time dependency is included, the function $\boldsymbol{E}(t + dt) = \boldsymbol{V}(\boldsymbol{S}(t))$ may be computed by a set of equations of the form

$$e(j, t + dt) = (k\, d/dt + 1) \sum_i s(i.t) w(i, j) \tag{7}$$

where $e(j, t)$ is the value of the $j$th value state-variable in the vector $\boldsymbol{E}$ at time $t$ $s(i, t)$ is the value of the $i$th input variable at time $t$ $w(i, j)$ is a coefficient, or weight, that defines the contribution of $s(i)$ to $e(j)$.

Each individual may have a different set of "values", i.e., a different weight matrix in its value judgment function $\boldsymbol{V}$.

The factor $(kd/dt + 1)$ indicates that a value judgment is typically dependent on the temporal derivative of its input variables as well as on their steady-state values. If $k > 1$, then the rate of change of the input factors becomes more important than their absolute values. For $k > 0$, need reduction and escape from pain are rewarding. The more rapid the escape, the more intense, but short-lived, the reward.

Formula (8) suggests how a $VJ$ function might compute the value state-variable "happiness":

$$\begin{aligned} \text{happiness} = (k\, d/dt + 1)(&\text{success-expectation} \\ &+ \text{hope-frustration} \\ &+ \text{love-hate} \\ &+ \text{comfort-fear} \\ &+ \text{joy-despair}) \end{aligned} \tag{8}$$

where success, hope, love, comfort, joy are all positive value state-variables that contribute to happiness, and expectation, frustration, hate, fear, and despair are all negative value state-variables that tend to reduce or diminish happiness. In this example, the plus and minus signs result from $+1$ weights assigned to the positive-value state-variables, and $-1$ weights assigned to the negative-value state-variables. Of course, different brains may assign different values to these weights.

Expectation is listed in formula (8) as a negative state-variable because the positive contribution of success is diminished if success_observed does not meet or exceed success_expected. This suggests that happiness could be increased if expectations were lower. However, when $k > 0$, the hope reduction that accompanies expectation downgrading may be just as punishing as the disappointments that result from unrealistic expectations, at least in the short term. Therefore, lowering expectations is a good strategy for increasing happiness only if expectations are lowered very slowly, or are already low to begin with.

Fig. 19 shows an example of how a VJ module might compute pleasure-pain. Skin and muscle are known to contain arrays of pain sensors that detect tissue damage. Specific receptors for pleasure are not known to exist, but pleasure state-variables can easily be computed from intermediate state-variables that are computed directly from skin sensors.

The VJ module in Fig. 19 computes "pleasure-pain" as a function of the intermediate state-variables of "softness", "warmth", and "gentle stroking of the skin". These intermediate state-variables are computed by low level SP modules.

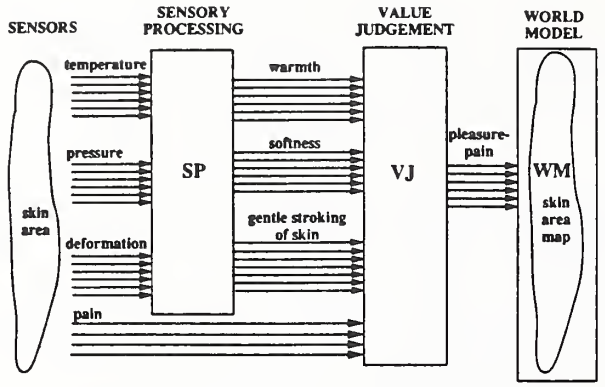



Fig. 19. How a VJ value judgment module might evaluate tactile and thermal sensory input. In this example, pleasure–pain is computed by a VJ module as a function of "warmth," "softness," and "gentle stroking" state-variables recognized by an SP module, plus inputs directly from pain sensors in the skin. Pleasure–pain value state-variables are assigned to pixel frames of the world model map of the skin area.

"warmth" is computed from temperature sensors in the skin. "softness" is computed as a function of "pressure" and "deformation" (i.e., stretch) sensors. "gentle stroking of the skin" is computed by a spatiotemporal analysis of skin pressure and deformation sensor arrays that is analogous to image flow processing of visual information from the eyes. Pain sensors go directly from the skin area to the VJ module.

In the processing of data from sensors in the skin, all of the computations preserve the topological mapping of the skin area. Warmth is associated with the area in which the temperature sensors are elevated. Softness is associated with the area where pressure and deformation are in the correct ratio. Gentle stroking is associated with the area in which the proper spatiotemporal patterns of pressure and deformation are observed. Pain is associated with the area where pain sensors are located. Finally, pleasure-pain is associated with the area from which the pleasure-pain factors originate. A pleasure-pain state-variable can thus be assigned to the knowledge frames of the skin pixels that lie within that area.

### D. Value State-Variable Map Overlays

When objects or regions of space are projected on a world map or egosphere, the value state-variables in the frames of those objects or regions can be represented as overlays on the projected regions. When this is done, value state-variables such as comfort, fear, love, hate, danger, and safe will appear overlaid on specific objects or regions of space. BG modules can then perform path planning algorithms that steer away from objects or regions overlaid with fear, or danger, and steer toward or remain close to those overlaid with attractiveness, or comfort. Behavior generation may generate attack commands for target objects or persons overlaid with hate. Protect, or care-for, commands may be generated for target objects overlaid with love.

Projection of uncertainty, believability, and importance value state-variables on the egosphere enables BG modules to perform the computations necessary for manipulating sensors and focusing attention.

Confidence, uncertainty, and hope state-variables may also be used to modify the effect of other value judgments. For example, if a task goal frame has a high hope variable but low confidence variable, behavior may be directed toward the hoped-for goal, but cautiously. On the other hand, if both hope and confidence are high, pursuit of the goal may be much more aggressive.

The real-time computation of value state-variables for varying task and world model conditions provides the basis for complex situation dependent behavior [56].

## XVI. Neural Computation

*Theorem:* All of the processes described previously for the BG, WM, SP, and VJ modules, whether implicit or explicit, can be implemented in neural net or connectionist architectures, and hence could be implemented in a biological neuronal substrate.

Modeling of the neurophysiology and anatomy of the brain by a variety of mathematical and computational mechanisms has been discussed in a number of publications [16], [27], [34], [35] ,[55], [59]–[64]. Many of the submodules in the BG, WM, SP, and VJ modules can be implemented by functions of the form $\boldsymbol{P}=\boldsymbol{H}(\boldsymbol{S})$. This type of computation can be accomplished directly by a typical layer of neurons that might make up a section of cortex or a subcortical nucleus.

To a first approximation, any single neuron, such as illustrated in Fig. 20, can compute a linear single valued function of the form

$$p(k) = h(\boldsymbol{S}) = \sum_{i=1}^{N} s(i)w(i,k) \tag{9}$$

where $p(k)$ is the output of the $k$th neuron; $\boldsymbol{S} = (s(1).s(2)....s(i)....,s(N))$ is an ordered set of input variables carried by input fibers defining an input vector; $\boldsymbol{W} = (w(1,k), w(2,k),...w(i,k),...w(N,k)$ is an ordered set of synaptic weights connecting the $N$ input fibers to the $k$th neuron; and $h(\boldsymbol{S})$ is the internal product between the input vector and the synaptic weight vector.

A set of neurons of the type illustrated in Fig. 20 can therefore compute the vector function

$$\boldsymbol{P} = \boldsymbol{H}(\boldsymbol{S}) \tag{10}$$

where $\boldsymbol{P} = (p(1), p(2)....p(k)....p(L))$ is an ordered set of output variables carried by output fibers defining an output vector.

Axon and dendrite interconnections between layers, and within layers, can produce structures of the form illustrated in Fig. 4. State driven switching functions produce structures such as illustrated in Figs. 2 and 3. It has been shown how such structures can produce behavior that is sensory-interactive, goal-directed, and value driven.

The physical mechanisms of computation in a neuronal computing module are produced by the effect of chemical activation on synaptic sites. These are analog parameters with time constants governed by diffusion and enzyme activity rates. Computational time constants can vary from milliseconds to minutes, or even hours or days, depending on the chemicals

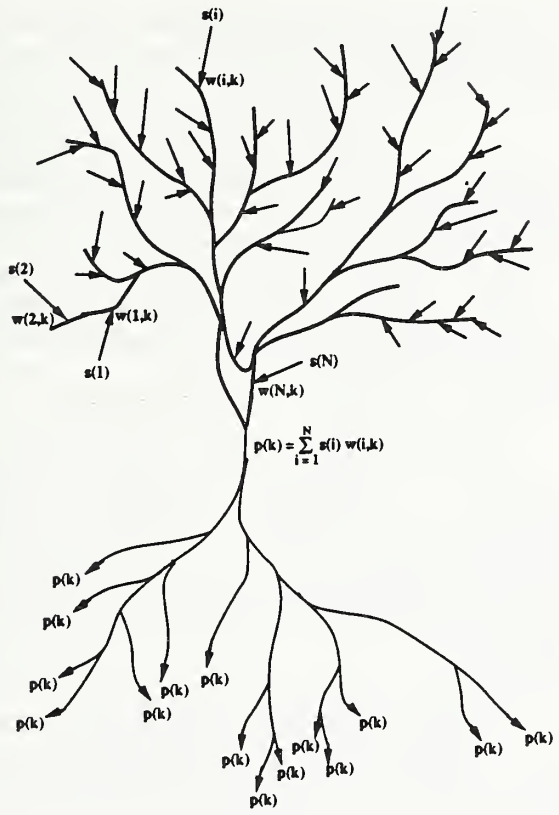



Fig. 20. A neuron computes the scalar value $p(k)$ as the inner product of the input vector $s(1), s(2), \ldots, s(i), \ldots, s(N)$ and the weight vector $w(1,k), w(2,k), \ldots w(i,k), \ldots, w(N,k)$.

carrying the messages, the enzymes controlling the decay time constants, the diffusion rates, and the physical locations of neurological sites of synaptic activity.

The time dependent functional relationship between input fiber firing vector $\boldsymbol{S}(t)$ and the output cell firing vector $\boldsymbol{P}(t)$ can be captured by making the neural net computing module time dependent

$$\boldsymbol{P}(t+dt) = \boldsymbol{H}(\boldsymbol{S}(t)). \tag{11}$$

The physical arrangement of input fibers in Fig. 20 can also produce many types of nonlinear interactions between input variables. It can, in fact, be shown that a computational module consisting of neurons of the type illustrated in Fig. 20 can compute any single valued arithmetic, vector, or logical function, IF/THEN rule, or memory retrieval operation that can be represented in the form $\boldsymbol{P}(t+dt) = \boldsymbol{H}(\boldsymbol{S}(t))$. By interconnecting $\boldsymbol{P}(t+dt) = \boldsymbol{H}(\boldsymbol{S}(t))$ computational modules in various ways, a number of additional important mathematical operations can be computed, including finite state automata, spatial and temporal differentiation and integration, tapped delay lines, spatial and temporal auto- and crosscorrelation, coordinate transformation, image scrolling and warping, pattern recognition, content addressable memory, and sampled-data, state-space feedback control. [59]–[63].

In a two layer neural net such as a Perceptron, or a brain model such as CMAC [27], [34], [35], the nonlinear function

$$\boldsymbol{P}(t+dt) = \boldsymbol{H}(\boldsymbol{S}(t))$$

is computed by a pair of functions

$$\boldsymbol{A}(\tau) = \boldsymbol{F}(\boldsymbol{S}(t)) \tag{12}$$

$$\boldsymbol{P}(t+dt) = \boldsymbol{G}(\boldsymbol{A}(\tau)) \tag{13}$$

where $\boldsymbol{S}(t)$ represents a vector of firing rates $s(i,t)$ on a set of input fibers at time $t$, $\boldsymbol{A}(\tau)$ represents a vector of firing rates $a(j,\tau)$ of a set of association cells at time $\tau = t + dt/2$, $\boldsymbol{P}(t+dt)$ represents a vector of firing rates $p(k, t+dt)$ on a set of output fibers at time $t+dt$, $F$ is the function that maps $S$ into $\boldsymbol{A}$, and $\boldsymbol{G}$ is the function that maps $\boldsymbol{A}$ into $\boldsymbol{P}$.

The function $\boldsymbol{F}$ is generally considered to be fixed, serving the function of an address decoder (or recoder) that transforms the input vector $\boldsymbol{S}$ into an association cell vector $\boldsymbol{A}$. The firing rate of each association cell $a(j,t)$ thus depends on the input vector $\boldsymbol{S}$ and the details of the interconnecting matrix of interneurones between the input fibers and association cells that define the function $\boldsymbol{F}$. Recoding from $\boldsymbol{S}$ to $\boldsymbol{A}$ can enlarge the number of patterns that can be recognized by increasing the dimensionality of the pattern space, and can permit the storage of nonlinear functions and the use of nonlinear decision surfaces by circumscribing the neighborhood of generalization. [34], [35].

The function $\boldsymbol{G}$ depends on the values of a set of synaptic weights $w(j,k)$ that connect the association cells to the output cells. The value computed by each output neuron $p(k,t)$ at time $t$ is

$$p(k, t+dt) = \sum_j a(j) w(j.k) \tag{14}$$

where $w(j,k)$=synaptic weight from $a(j)$ to $p(k)$.

The weights $w(j,k)$ may be modified during the learning process so as to modify the function $\boldsymbol{G}$, and hence the function $\boldsymbol{H}$.

Additional layers between input and output can produce indirect addressing and list processing functions, including tree search and relaxation processes [16], [61]. Thus, virtually all of the computational functions required of an intelligent system can be produced by neuronal circuitry of the type known to exist in the brains of intelligent creatures.

## XVII. Learning

It is not within the scope of this paper to review of the field of learning. However, no theory of intelligence can be complete without addressing this phenomenon. Learning is one of several processes by which world knowledge and task knowledge become embedded in the computing modules of an intelligent system. In biological systems, knowledge is also provided by genetic and growth mechanisms. In artificial systems, knowledge is most often provided through the processes of hardware engineering and software programming.

In the notation of (13), learning is the process of modifying the $\boldsymbol{G}$ function. This in turn, modifies the $\boldsymbol{P} = \boldsymbol{H}(\boldsymbol{S})$ functions that reside in BG, WM, SP, and VJ modules. Thus through learning, the behavior generation system can acquire new behavioral skills, the world model can be updated, the

sensory processing system can refine its ability to interpret sensory input, and new parameters can be instilled in the value judgment system.

The change in strength of synaptic weights $w(j,k)$ wrought by the learning process may be described by a formula of the form

$$dw(j,k,t) = g(t)a(j,t)p(k,t) \tag{15}$$

where $dw(j,k,t)$ is the change in the synaptic weight $w(j,k,t)$ between $t$ and $t+dt$; $g(t)$ is the learning gain at time $t$; $a(j,t)$ is the firing rate of association cell $j$ at time $t$; and $p(k,t)$ is the firing rate of output neuron $k$ at time $t$.

If $g(t)$ is positive, the effect will be to reward or strengthen active synaptic weights. If $g(t)$ is negative, the effect will be to punish, or weaken active synaptic weights.

After each learning experience, the new strength of synaptic weights is given by

$$w(j,k,t+dt) = w(j,k,t) + dw(j,k,t). \tag{16}$$

### A. Mechanisms of Learning

Observations from psychology and neural net research suggests that there are at least three major types of learning: repetition, reinforcement, and specific error correction learning.

*1) Repetition:* Repetition learning occurs due to repetition alone, without any feedback from the results of action. For this type of learning, the gain function g is a small positive constant. This implies that learning takes place solely on the basis of coincidence between presynaptic and postsynaptic activity. Coincident activity strengthens synaptic connections and increases the probability that the same output activity will be repeated the next time the same input is experienced.

Repetition learning was first hypothesized by Hebb, and is sometimes called Hebbian learning. Hebb hypothesized that repetition learning would cause assemblies of cells to form associations between coincident events, thereby producing conditioning. Hebbian learning has been simulated in neural nets, with some positive results. However, much more powerful learning effects can be obtained with reinforcement learning.

*2) Reinforcement:* Reinforcement learning incorporates feedback from the results of action. In reinforcement learning, the learning gain factor $g(t)$ varies with time such that it conveys information as to whether the evaluation computed by the VJ module was good (rewarding), or bad (punishing). $g(t)$ is thus computed by a VJ function of the form

$$g(t+dt) = V(S(t)) \tag{17}$$

where $S(t)$ is a time dependent state vector defining the object, event, or region of space being evaluated.

For task learning

$$g(t+dt) = V\{R(t) - R_d(t)\} \tag{18}$$

where $R(t)$ is the actual task results at time $t$, $R_d(t)$ is the desired task results at time $t$, $R(t) - R_d(t)$ is the difference between the actual results and the desired results.

Task learning may modify weights in BG modules that define parameters in subtasks, or the weights that define decision functions in BG state-tables, or the value of state-variables in the task frame, such as task priority, expected cost, risk, or benefit. Task learning may thus modify both the probability that a particular task will be selected under certain conditions, and the way that the task is decomposed and executed when it is selected.

Attribute learning modifies weights that define state-variables in the attribute list of entity or event frames in the world model. Attribute learning was described earlier by (3) and (4).

For attribute learning

$$g(t+dt) = K_s(i,t)[1 - K_m(j,t)]V(\text{attribute}_j) \tag{19}$$

where $K_s(i,t)$ is the degree of confidence in the sensory observation of the $i$th real world attribute at time $t$ (See formula (4)); $K_m(j,t)$ is the degree of confidence in the prediction of the $j$th world model attribute at time $t$; and $V(\text{attribute}_j)$ is the importance of the $j$th world model attribute.

In general, rewarding reinforcement causes neurons with active synaptic inputs to increase the value or probability of their output the next time the same situation arises, or through generalization to increase the value or probability of their output the next time almost-the-same situation arises. Every time the rewarding situation occurs, the same synapses are strengthened, and the output (or its probability of occurring) is increased further.

For neurons in the goal selection portion of the BG modules, the rewarding reinforcement causes rewarding goals to be selected more often. Following learning, the probabilities are increased of EX submodules selecting next-states that were rewarded during learning. Similarly, the probabilities are increased of PL and JA submodules selecting plans that were successful, and hence rewarding, in the past.

For neurons in the WM modules, rewarding results following an action causes reward expectations to be stored in the frame of the task being executed. This leads to reward values being increased on nodes in planning graphs leading up to the rewarding results. Cost/benefit values placed in the frames of objects, events, and tasks associated with the rewarding results are also increased. As a result, the more rewarding the result of behavior, the more the behavior tends to be repeated.

Reward reinforcement learning in the BG system is a form of positive feedback. The more rewarding the task, the greater the probability that it will be selected again. The more it is selected, the more reward is produced and the more the tendency to select it is increased. This can drive the goal selection system into saturation, producing effects like addiction, unless some other process such as fatigue, boredom, or satiety produce a commensurate amount of negative $g(t)$ that is distributed over the population of weights being modified.

Punishing reinforcement, or error correcting, learning occurs when $g(t)$ is negative, i.e., punishing. In biological brains, error correction weakens synaptic weights that are active immediately prior to punishing evaluations from the emotional

system. This causes the neurons activated by those synapses to decrease their output the next time the same situation arises. Every time the situation occurs and the punishing evaluation is given, the same synapses are weakened and the output (or its probability of occurring) is reduced.

For neurons in the goal selection portion of the BG modules, error correction tends to cause punishing tasks to be avoided. It decreases the probability of EX submodules selecting a punishing next state. It decreases the probability of PL and JA submodules selecting a punishing plan.

For neurons in the WM modules, punishment observed to follow an action causes punishment state variables to be inserted into the attribute list of the tasks, objects, events, or regions of space associated with the punishing feedback. Thus, punishment can be expected the next time the same action is performed on the same object, or the same event is encountered, or the same region of space is entered. Punishment expectations (i.e., fear) can be placed in the nodes of planning graphs leading to punishing task results. Thus, the more punishing the task, the more the task tends to be avoided.

Error correction learning is a form of negative feedback. With each training experience, the amount of error is reduced, and hence the amount of punishment. Error correction is therefore self limiting and tends to converge toward a stable result. It produces no tendencies toward addiction.

It does, however, reduce the net value of the synaptic weight pool. Without some other process such as excitement, or satisfaction, to generate a commensurate amount of reward reinforcement, there could result a reduction in stimulus to action, or lethargy.

*3) Specific Error Correction Learning:* In specific error correction, sometimes called teacher learning, not only is the overall behavioral result $g(t)$ known, but the correct or desired response $p_d(k,t)$ of each output neuron is provided by a teacher. Thus, the precise error $(p(k)-p_d(k))$ for each neuron is known. This correction can then be applied specifically to the weights of each neuron in an amount proportional to the direction and magnitude of the error of that neuron. This can be described by

$$dw(j,k,t) = g(t)a(j,t)(p(k,t) - p_d(k,t)) \qquad (20)$$

where $p_d(k,t)$ is the desired firing rate of neuron $k$ at $t$ and $-1 \leq g(t) < 0$.

Teacher learning tends to converge rapidly to stable precise results because it has knowledge of the desired firing rate for each neuron. Teacher learning is always error correcting. The teacher provides the correct response, and anything different is an error. Therefore, $g(t)$ must always be negative to correct the error. A positive $g(t)$ would only tend to increase the error.

If the value of $g(t)$ is set to $-1$, the result is one-shot learning. One-shot learning is learning that takes only one training cycle to achieve perfect storage and recall. One-shot teacher learning is often used for world model map and entity attribute updates. The SP module produces an observed value for each pixel, and this becomes the desired value to be stored in a world model map. A SP module may also produce observed values for entity attributes. These become desired values to be stored in the world model entity frame.

Teacher learning may also be used for task skill learning in cases where a high level BG module can act as a teacher to a lower level BG module, i.e., by providing desired output responses to specific command and feedback inputs.

It should be noted that, even though teacher learning may be one-shot, task skill learning by teacher may require many training cycles, because there may be very many ways that a task can be perturbed from its ideal performance trajectory. The proper response to all of these must be learned before the task skill is fully mastered. Also, the teacher may not have full access to all the sensory input going to the BG module that is being taught. Thus, the task teacher may not always be fully informed, and therefore may not always generate the correct desired responses.

Since teacher learning is punishing, it must be accompanied by some reward reinforcement to prevent eventually driving synaptic weights to zero. There is some evidence, that both reward reinforcement, and teacher learning, take place simultaneously in the cerebellum. Reward signals are thought to be carried by diffuse noradrenergic fibers that affect many thousands of neurons in the same way, while error correction signals are believed to be carried by climbing fibers each of which specifically targets a single neuron or a very small groups of neurons [27].

It should be noted, however, that much of the evidence for neuronal learning is ambiguous, and the exact mechanisms of learning in the brain are still uncertain. The very existence of learning in particular regions of the brain (including the cerebellum) is still controversial [65]. In fact, most of the interesting questions remain unanswered about how and where learning occurs in the neural substrate, and how learning produces all the effects and capabilities observed in the brain.

There are also many related questions as to the relationships between learning, instinct, imprinting, and the evolution of behavior in individuals and species.

## XVIII. Conclusion

The theory of intelligence presented here is only an outline. It is far from complete. Most of the theorems have not been proven. Much of what has been presented is hypothesis and argument from analogy. The references cited in the bibliography are by no means a comprehensive review of the subject, or even a set of representative pointers into the literature. They simply support specific points. A complete list of references relevant to a theory of intelligence would fill a volume of many hundreds of pages. Many important issues remain uncertain and many aspects of intelligent behavior are unexplained.

Yet, despite its incomplete character and hypothetical nature, the proffered theory explains a lot. It is both rich and self consistent, but more important, it brings together concepts from a wide variety of disciplines into a single conceptual framework. There is no question of the need for a unifying theory. The amount of research currently underway is huge, and progress is rapid in many individual areas. Unfortunately, positive results in isolated fields of research have not coalesced into commensurate progress toward a general understanding of

the nature of intelligence itself, or even toward improved abilities to build intelligent machine systems. Intelligent systems research is seriously impeded because of the lack of a widely accepted theoretical framework. Even a common definition of terms would represent a major step forward.

The model presented here only suggests how the neural substrate could generate the phenomena of intelligence, and how computer systems might be designed so as to produce intelligent behavior in machines. No claim is made that the proposed architecture fully explains how intelligence actually is generated in the brain. Natural intelligence is almost certainly generated in a great variety of ways, by a large number of mechanisms. Only a few of the possibilities have been suggested here.

The theory is expressed almost entirely in terms of explicit representations of the functionality of BG, WM, SP, and VJ modules. This almost certainly is not the way the brains of lower forms, such as insects, generate intelligent behavior. In simple brains, the functionality of planning, representing space, modeling and perceiving entities and events is almost surely represented implicitly, embedded in the specific connectivity of neuronal circuitry, and controlled by instinct.

In more sophisticated brains, however, functionality most likely is represented explicitly. For example, spatial information is quite probably represented in world and egosphere map overlays, and map pixels may indeed have frames. One of the principal characteristics of the brain is that the neural substrate is arranged in layers that have the topological properties of maps. Output from one layer of neurons selects, or addresses, sets of neurons in the next. This is a form a indirect addressing that can easily give rise to list structures, list processing systems, and object-oriented data structures. Symbolic information about entities, events, and tasks may very well be represented in neuronal list structures with the properties of frames. In some instances, planning probably is accomplished by searching game graphs, or by invoking rules of the form IF (S)/THEN (P).

Implicit representations have an advantage of simplicity, but at the expense of flexibility. Implicit representations have difficulty in producing adaptive behavior, because learning and generalization take place only over local neighborhoods in state-space. On the other hand, explicit representations are complex, but with the complexity comes flexibility and generality. Explicitly represented information is easily modified, and generalization can take place over entire classes of entities. Class properties can be inherited by subclasses, entity attributes can be modified by one-shot learning, and small changes in task or world knowledge can produce radically altered behavior. With explicit representations of knowledge and functionality, behavior can become adaptive, even creative.

This paper attempts to outline an architectural framework that can describe both natural and artificial implementations of intelligent systems. Hopefully, this framework will stimulate researchers to test its hypotheses, and correct its assumptions and logic where and when they are shown to be wrong. The near term goal should be to develop a theoretical model with sufficient mathematical rigor to support an engineering science of intelligent machine systems. The long term goal should be a full understanding of the nature of intelligence and behavior in both artificial and natural systems.

## ACKNOWLEDGMENT

The author wishes to thank Alex Meystel, Robert Rosenfeld, John Simpson, Martin Herman, Ron Lumia, and Rick Quintero for their numerous helpful comments, and Cheryl Hutchins for her help in preparation of this manuscript. Funding to support this research has been derived from DARPA, NIST, NASA, Army, Navy, Air Force, and Bureau of Mines sources.

## REFERENCES

[1] L. D. Erman, F. Hayes-Roth, V. R. Lesser, and D. R. Reddy, D. R., "Hearsay-II speech understanding system: Integrating knowledge to resolve uncertainty," *Computer Survey*, vol. 23, pp. 213–253, June 1980.
[2] J. E. Laird, A. Newell, and P. Rosenbloom, "SOAR: An architecture for general intelligence," *Artificial Intell.*, vol. 33, pp. 1–64, 1987.
[3] Honeywell, Inc., "Intelligent Task Automation Interim Tech. Rep. II-4", Dec. 1987.
[4] J. Lowerie *et al.*, "Autonomous land vehicle," Annu. Rep., ETL-0413, Martin Marietta Denver Aerospace, July 1986.
[5] D. Smith and M. Broadwell, "Plan coordination in support of expert systems integration," in *Knowledge-Based Planning Workshop Proc.*, Austin, TX, Dec. 1987.
[6] J. R. Greenwood, G. Stachnick, H. S. Kaye, "A procedural reasoning system for army maneuver planning," in *Knowledge-Based Planning Workshop Proc.*, Austin, TX, Dec. 1987.
[7] A. J. Barbera, J. S. Albus, M. L. Fitzgerald, and L. S. Haynes, "RCS: The NBS real-time control system," in *Proc. Robots 8 Conf. Exposition*, Detroit, MI, June 1984.
[8] R. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robotics Automat.*, vol. RA-2, Mar. 1986.
[9] G. N. Saridis, "Foundations of the theory of intelligent controls," in *Proc. IEEE Workshop on Intelligent Contr.*, 1985.
[10] A. Meystel, "Intelligent control in robotics," *J. Robotic Syst.*, 1988.
[11] J. A. Simpson, R. J. Hocken, and J. S. Albus, "The Automated manufacturing research facility of the National Bureau of Standards," *J. Manufact. Syst.*, vol. 1, no. 1, 1983.
[12] J. S. Albus, C. McLean, A. J. Barbera, and M. L. Fitzgerald, "An architecture for real-time sensory-interactive control of robots in a manufacturing environment," presented at the 4th IFAC/IFIP Symp. on Inform. Contr. Problems in Manufacturing Technology, Gaithersburg, MD, Oct. 1982.
[13] J. S. Albus, "System description and design architecture for multiple autonomous undersea vehicles," Nat. Inst. Standards and Tech., Tech. Rep. 1251, Gaithersburg, MD, Sept. 1988.
[14] J. S. Albus, H. G. McCain, and R. Lumia, "NASA/NBS standard reference model for telerobot control system architecture (NASREM)" Nat. Inst. Standards and Tech., Tech. Rep. 1235, Gaithersburg, MD, 1989.
[15] B. Hayes-Roth, "A blackboard architecture for control," *Artificial Intell.*, pp. 252–321, 1985.
[16] J. S. Albus, *Brains, Behavior, and Robotics*. Peterbourough, NH: BYTE/McGraw-Hill, 1981.
[17] G. A. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information," *Psych. Rev.*, vol. 63, pp. 71–97, 1956.
[18] A. Meystel, "Theoretical foundations of planning and navigation for autonomous robots," *Int. J. Intelligent Syst.*, vol. 2, pp. 73–128, 1987
[19] M. Minsky, "A framework for representing knowledge," in *The Psychology of Computer Vision*, P. Winston, Ed. New York: McGraw-Hill, 1975, pp. 211–277.
[20] E. D. Sacerdoti, *A Structure for Plans and Behavior*. New York: Elsevier, 1977.
[21] R. C. Schank and R. P. Abelson, *Scripts Plans Goals and Understanding* Hillsdale, NJ: Lawrence Erlbaum, 1977.
[22] D. M. Lyons and M. A. Arbib, "Formal model of distributed computation sensory based robot control," *IEEE J. Robotics and Automat. Rev.*, 1988.
[23] D. W. Payton, "Internalized plans: A representation for action resources," *Robotics and Autonomous Syst.*, vol. 6, pp. 89–103, 1990.
[24] A. Sathi and M. Fox, "Constraint-directed negotiation of resource reallocations," CMU-RI-TR-89-12, Carnegie Mellon Robotics Institute Tech. Rep., Mar., 1989

[25] V. B. Brooks, *The Neural Basis of Motor Control*. Oxford, UK: Oxford Univ. Press, 1986.
[26] J. Piaget, *The Origins of Intelligence in Children*. New York: Int. Universities Press, 1952.
[27] J. S. Albus, "A theory of cerebellar function," *Math. Biosci.*, vol. 10, pp. 25–61, 1971.
[28] P. D. MacLean, *A Triune Concept of the Brain and Behavior*. Toronto, ON: Univ. Toronto Press, 1973.
[29] A. Schopenhauer, "The World As Will and Idea", 1883, in *The Philosophy of Schopenhauer*, Irwin Edman, Ed. Ithaca, NY: New York: Random House, 1928.
[30] J. J. Gibson, *The Ecological Approach to Visual Perception*. Ithaca, NY: Cornell Univ. Press, 1966.
[31] D. H. Hubel and T. N. Wiesel, "Ferrier lecture: Functional architecture of macaque monkey visual cortex," *Proc. Roy. Soc. Lond. B*. vol. 198, 1977, pp. 1–59.
[32] H. Samet, "The quadtree and related hierarchical data structures," *Computer Surveys*, pp. 16–2, 1984.
[33] P. Kinerva, *Sparse Distributed Memory*. Cambridge, MA: MIT Press, 1988.
[34] J. S. Albus, "A new approach to manipulator control: The cerebellar model articulation controller (CMAC)," *Trans. ASME*, Sept. 1975.
[35] ——, "Data storage in the cerebellar model articulation controller (CMAC)," *Trans. ASME*, Sept. 1975.
[36] M. Bradey, "Computational approaches to image understanding," *ACM Comput. Surveys*, vol. 14, Mar. 1982.
[37] T. Binford, "Inferring surfaces from images," *Artificial Intell.*, vol. 17, pp. 205–244, 1981.
[38] D. Marr and H. K. Nishihara. "Representation and recognition of the spatial organization of three-dimensional shapes," in *Proc. Roy. Soc. Lond. B*, vol. 200, pp. 269–294, 1978.
[39] R. F. Riesenfeld, "Applications of B-spline approximation to geometric problems of computer aided design," Ph.D. dissertation, Syracuse Univ. 1973. available at Univ. Utah, UTEC -CSc-73-126.
[40] J. J. Koenderink, "The structure of images," *Biolog. Cybern.*, vol. 50, 1984.
[41] J. C. Pearson, J. Gelfand, W. Sullivan, R. Peterson, and C. Spence, "A neural network approach to sensory fusion," in *Proc. SPIE Sensor Fusion Conf.*, Orlando, FL, 1988.
[42] D. L. Sparks and M. Jay, "The role of the primate superior colliculus in sensorimotor integration," in *Vision, Brain, and Cooperative Computation*, Arbib and Hanson, Eds. Cambridge, MA: MIT Press, 1987.
[43] R. A. Andersen and D. Zipser, "The role of the posterior parietal cortex in coordinate transformations for visual-motor integration," *Can. J. Physiol. Pharmacol.*, vol. 66, pp. 488–501, 1988.
[44] D. Marr, *Vision*. San Francisco,CA: Freeman, 1982.
[45] J. S. Albus and T. H. Hong, "Motion, Depth, and Image Flow", in *Proc. IEEE Robotics and Automation*, Cincinnati, OH, 1990 (in process).
[46] D. Raviv and J. S. Albus, "Closed-form massively-parallel range-from-image flow algorithm," NISTIR 4450, National Inst. of Standards & Technology, Gaithersburg, MD, 1990.
[47] E. Kent and J. S. Albus, "Servoed world models as interfaces between robot control systems and sensory data," *Robotica*, vol. 2, pp. 17–25, 1984.
[48] E. D. Dickmanns and T. H. Christians, "Relative 3D-state estimation for autonomous visual guidance of road vehicles," *Intelligent Autonomous Syst.*, vol. 2, Amsterdam, The Netherlands, Dec. 11–14, 1989.
[49] R. Bajcsy, "Passive perception vs. active perception" in *Proc. IEEE Workshop on Computer Vision*, Ann Arbor, MI, 1986.
[50] K. Chaconas and M. Nashman, "Visual perception processing in a hierarchical control system: Level 1," Nat. Inst. Standards Technol. Tech. Note 1260, June 1989.
[51] Y. L. Grand, *Form and Space Vision*, Table 21, Ind. Univ. Press, Bloomington, IN, 1967.
[52] A. L. Yarbus, *Eye Movements and Vision*. New York: Plenum, 1967
[53] D. C. Van Essen, "Functional organization of primate visual cortex," *Cerebral Cortex*, vol. 3, A. Peters and E. G. Jones, Eds. New York: Plenum, 1985, pp. 259–329.
[54] J. H. R. Maunsell and W. T. Newsome, "Visual processing in monkey extrastriate cortex," *Ann. Rev. Neurosci.*, vol. 10, pp. 363–401, 1987.
[55] S. Grossberg, *Studies of Mind and Brain*. Amsterdam: Reidel, 1982.
[56] G. E. Pugh, *The Biological Origin of Human Values*. New York: Basic Books, 1977.
[57] A. C. Guyton, *Organ Physiology, Structure and Function of the Nervous System*, second ed. Philadelphia, PA: Saunders, 1976.
[58] W. B. Scoville and B. Milner, "Loss of recent memory after bilateral hippocampal lesions," *J. Neurophysiol. Neurosurgery Psychiatry*, vol. 20, no. 11, pp. 11–29, 1957.
[59] J. S. Albus, "Mechanisms of planning and problem solving in the brain," *Math. Biosci.*, vol. 45, pp. 247–293, 1979.
[60] S. Grossberg, Ed., *Neural Networks and Natural Intelligence*. Cambridge, MA: Bradford Books—MIT Press, 1988.
[61] J. S. Albus, "The cerebellum: A substrate for list-processing in the brain," in *Cybernetics, Artificial Intelligence and Ecology*, H. W. Robinson and D. E. Knight, Eds. Washington, DC: Spartan Books, 1972.
[62] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," in *Proc. Nat. Acad. Sci.*, vol. 79, 1982, pp. 2554–2558.
[63] B. Widrow and R. Winter, "Neural nets for adpative filtering and adaptive pattern recognition," *Comput.*, vol. 21, no. 3, 1988.
[64] M. Minsky and S. Papert, *An Introduction to Computational Geometry*. Cambridge, MA: MIT Press, 1969.
[65] M. Ito, *The Cerebellum and Neuronal Control*. New York: Raven, 1984, ch. 10.

**James S. Albus** received the B. S. degree is physics in 1957 from Wheaton College, Wheaton, IL, the M.S. degree in electrical engineering in 1958 from Ohio State University, Columbus, and the Ph. D. degree in electrical engineering from the University of Maryland, College Park.

He is presently Chief of the Robot Systems Division, Laboartory for Manufacturing Engineering, National Institute of Standards and Technology, where he is responsible for robotics and automated manufacturing systems interface standards research. He designed the control system architecture for the Automated Manufacturing Research Facility. Previously, he worked from 1956 to 1973 for NASA Goddard Space Flight Center, where he designed electro-optical systems for more than 15 NASA spacecraft. For a short time, he served as program manager of the NASA Artificial Intelligence Program.

Dr. Albus has received several awards for his work in control theory, including the National Institute of Standards and Technology Applied Research Award, the Department of Commerce Gold and Silver Medals, The Industrial Research IR-100 award, and the Joseph F. Engelberger Award (which was presented at the International Robot Symposium in October 1984 by the King of Sweden. He has written two books, *Brains, Behavior, and Robotics* (Byte/McGraw Hill, 1981) and *Peoples' Capitalism: The Econimics of the Robot Revolutions* (New World Books, 1976). He is also the author of numerous scientific papers, journal articles, and official government studies, and has had articles published in *Scientific American, Omni, Byte*, and *Futurist*.

# A Reference Model Architecture for Intelligent Systems Design

James S. Albus
Robot Systems Division
Manufacturing Engineering Laboratory
National Institute of Standards and Technology

## Introduction

An outline for a theory of intelligence has been published [1]. It defines intelligence as the ability to act appropriately in an uncertain environment, where appropriate action is that which increases the probability of success, and success is the achievement of behavioral goals. The intelligent system acts so as to maximize probability of success and minimize probability of failure. Both goals and success criteria are generated in the environment external to the intelligent system. At a minimum, intelligence requires the abilities to sense the environment, make decisions, and control action. Higher levels of intelligence require the abilities to recognize objects and events, store and use knowledge about the world, and to reason about and plan for the future. Advanced forms of intelligence have the abilities to perceive and analyze, to plot and scheme, to choose wisely and plan successfully in a complex, competitive, hostile world. The amount of intelligence is determined by the computational power of the computing engine, the sophistication and elegance of algorithms, the amount and quality of information and values, and the efficiency and reliability of the system architecture. The amount of intelligence can grow through programming, learning, and evolution. Intelligence is the product of natural selection, wherein more successful behavior is passed on to succeeding generations of intelligent systems, and less successful behavior dies out. Natural selection is driven by competition between individuals within a group, and groups within the world.

The above theory of intelligence is expressed in terms of a reference model architecture for real-time intelligent control systems based on the RCS (Real-time Control System) [2]. RCS partitions the control problem into four basic elements: behavior generation (or task decomposition), world modeling, sensory processing, and value judgment. It clusters these elements into computational nodes that have responsibility for specific subsystems, and arranges these nodes in hierarchical layers such that each layer has characteristic functionality and timing. The RCS reference model architecture has a systematic regularity, and recursive structure that suggests a canonical form.

This paper is divided into seven sections. Section 1 describes the evolution of the RCS system through its various versions. Section 2 gives an example of RCS applied to a machining workstation application. Section 3 describes the timing of real-time task decomposition (planning and execution) and sensory processing(sampling and integration) at the various layers in the RCS hierarchy. Sections 4, 5, 6, and 7 define the functionality and contents of the Task Decomposition, World Modeling, Sensory Processing, and Value Judgment modules respectively.

## Section 1. Evolution of RCS

RCS has evolved through variety of versions over a number of years as understanding of the complexity and sophistication of intelligent behavior has increased. The first implementation was designed for sensory-interactive robotics by Barbera in the mid 1970's [3]. In RCS-1, the emphasis was on combining commands with sensory feedback so as to compute the proper response to every combination of goals and states. The application was to control a robot arm with a structured light vision system in visual pursuit tasks. RCS-1 was heavily influenced by

biological models such as the Marr-Albus model of the cerebellum [4], and the Cerebellar Model Arithmetic Computer (CMAC) [5]. CMAC can implement a function of the form

$$\text{Command}_{t+1}(i\text{-}1) = H_i(\text{Command}_t(i), \text{Feedback}_t(i))$$

where $H_i$ is a single valued function of many variables
i is an index indicating the hierarchical level
t is an index indicating time

CMAC thus became the reference model building block of RCS-1, as shown in Figure 1(a). A hierarchy of these building blocks was used to implement a hierarchy of behaviors such as observed by Tinbergen [6] and others. CMAC becomes a state-machine when some of its outputs are fed directly back to the input, so RCS-1 was implemented as a set of state-machines arranged in a hierarchy of control levels. At each level, the input command effectively selects a behavior that is driven by feedback in stimulus-response fashion. RCS-1 is thus similar in many respects to Brooks subsumption architecture [7], except that RCS selects behaviors before the fact through goals expressed in commands, rather than after the fact through subsumption.

The next generation, RCS-2, was developed by Barbera, Fitzgerald, Kent, and others for manufacturing control in the NIST Automated Manufacturing Research Facility (AMRF) during the early 1980's [8,9,10]. The basic building block of RCS-2 is shown in Figure 1(b). The H function remained a finite state machine state-table executor. The new feature of RCS-2 was the inclusion of the G function consisting of a number of sensory processing algorithms including structured light and blob analysis algorithms. RCS-2 was used to define an eight level hierarchy consisting of Servo, Coordinate Transform, E-Move, Task, Workstation, Cell, Shop, and Facility levels of control. Only the first six levels were actually built. Two of the AMRF workstations fully implemented five levels of RCS-2. The control system for the Army Field Material Handling Robot (FMR)[11] was also implemented in RCS-2, as were the Army TMAP and TEAM semi-autonomous land vehicle projects.

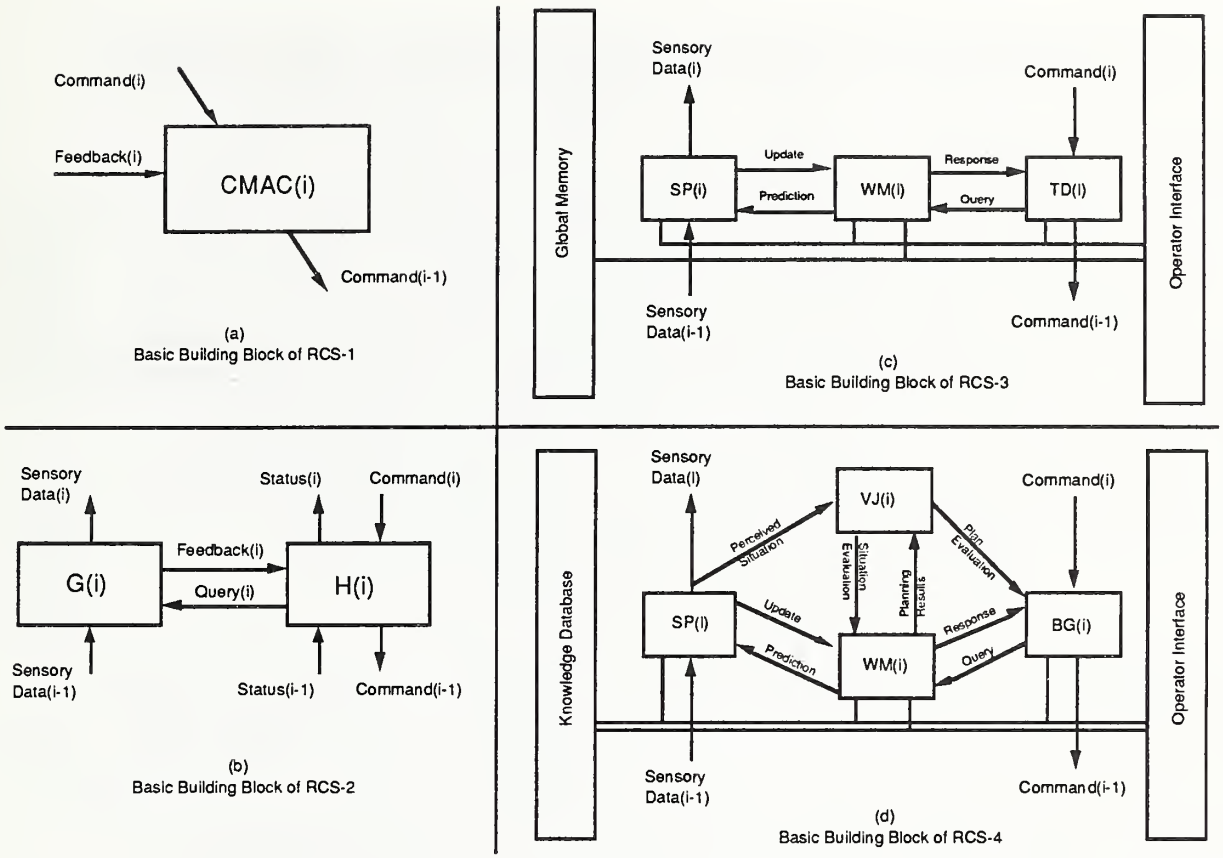



Figure 1. The basic building blocks of the RCS control paradigm.

RCS-3 was designed for the NBS/DARPA Multiple Autonomous Undersea Vehicle (MAUV) project [12] and was adapted for the NASA/NBS Standard Reference Model Telerobot Control System Architecture (NASREM) [13] that was used in the specification and design of the Next Generation Controller (NGC). The basic building block of RCS-3 is shown in Figure 1(c). The principle new features introduced in RCS-3 are the World Model and the operator interface. The inclusion of the World Model provides the basis for task planning and for model-based sensory processing. This led to refinement of the task decomposition (TD) modules so that each have a job assigner, and planner and executor for each of the subsystems assigned a job. This corresponds roughly to Saridis' three level control hierarchy [14].

RCS-4 is under current development by the NIST Robot Systems Division under Department of Commerce funding for fundamental research in Intelligent Machine Systems. The basic building block is shown in Figure 1(d). The principle new feature in RCS-4 is the explicit representation of the Value Judgment (VJ) system. VJ modules provide to the RCS-4 control system the type of functions provided to the biological brain by the limbic system. The VJ

modules contain processes that compute cost, benefit, and risk of planned actions; and that place value on objects, materials, territory, situations, events, and outcomes. Value state-variables define what goals are important, and what objects or regions should be attended-to, attacked, defended, assisted, or otherwise acted upon. Value judgments, or evaluation functions, are an essential part of any form of planning or learning. However, the evaluation functions are usually not made explicit, or assigned to a separate module as in RCS-4. The application of value judgments to intelligent control systems has been addressed by George Pugh [15]. The structure and function of VJ modules are developed more completely developed in [1].

RCS-4 also uses the term behavior generation (BG) in place of the RCS-3 term task decomposition (TD). The purpose of this change is to emphasize the degree of autonomous decision making. RCS-4 is designed to address highly autonomous applications in unstructured environments where high bandwidth communications are impossible, such as unmanned vehicles operating on the battlefield, deep undersea, or on distant planets. These applications require autonomous value judgments and sophisticated real-time perceptual capabilities. RCS-3 will continue to be used for less demanding applications such as manufacturing, construction, or telerobotics for near-space, or shallow undersea operations, where environments are more structured and communication bandwidth to a human interface is less restricted.

## Section 2. A Machining Workstation Example

Figure 3 illustrates how the RCS-3 system architecture can be applied to a specific machining workstation consisting of a machine tool, a robot, an inspection machine, and a part buffer. RCS-3 produces a layered graph of processing nodes, each of which contains a Task Decomposition (TD), World Modeling (WM), and Sensory Processing (SP) module. These modules are richly interconnected to each other by a communications system. At the lowest level, communications are typically implemented through common memory or message passing between processes on a single computer board. At middle levels, communications are typically implemented between multiple processors on a backplane bus. At high levels, communications can be implemented through bus gateways and local area networks. The global memory, or knowledge database, and operator interface are not shown in Figure 3.

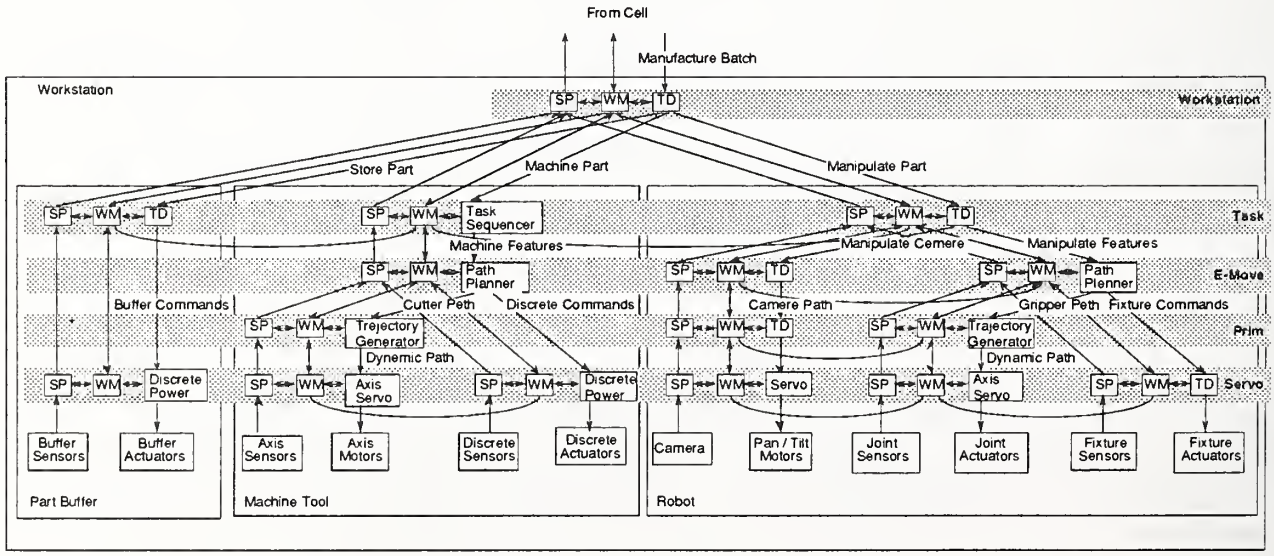


Figure 3. A RCS-3 application of a machining workstation containing a machine tool, part buffer, and robot with vision system.

Figure 3 illustrates the ability of RCS-3 to integrate discrete sensors such as microswitches with more complex sensors such as cameras and resolvers. Discrete commands can be issued to valves and fixtures, while continuous signals are provided to servoed actuators. Notice that in some branches of the control tree, nodes at some levels may be absent. For example, in the case of the part buffer, discrete commands at the Task level can be directly executed by the Servo level. In the case of the part fixture, discrete commands issued from the robot E-Move level can be executed by the Servo level. In these cases, the missing modules can be thought of as degenerate computing nodes that produce unity gain pass-through of inputs to outputs.

The branching of the control tree (for example, between the camera and manipulator subsystems of the robot), may depend on the particular algorithm chosen for decomposing a particular task. The specifications for branching reside in the task frame (defined later) of the current task being executed. Similarly, the specification for sharing information between WM modules at a level also are task dependent. In Figure 3, the horizontal curved lines represent the sharing of state information between subtrees in order to synchronize related tasks. The information that must be shared is also dependent on the specific choice of task decomposition algorithms defined in the task frame.

The functionality of each level in the control hierarchy can be derived from the characteristic timing of that level, and vice versa. For example, in a manufacturing environment, the following hierarchical levels are becoming more or less standard.

Level 7 -- Shop

The shop level schedules and controls the activities of one or more manufacturing cells for an extended period on the order of 24 hours. (The specific timing numbers given in this example are representative only, and may vary from application to application.) At the shop level, orders are sorted into batches and commands are issued to the cell level to develop a production schedule for each batch. At the shop level, the world model symbolic database contains names and attributes of orders and the inventory of tools and materials required to fill them. Maps describe the location of, and routing between, manufacturing cells.

Level 6—Cell

The cell level schedules and controls the activities of several workstations for about a one hour look ahead. Batches of parts and tools are scheduled into workstations, and commands are issued to workstations to perform machining, inspection, or material handling operations on batches or trays of parts. The world model symbolic database contains names and attributes of batches of parts and the tools and materials necessary to manufacture them. Maps describe the location of, and routing between, workstations. (The Shop and Cell levels are above the levels shown in the Figure 3 example.) The output from the cell level provides input to the workstation level.

Level 5—Workstation

The workstation level schedules tasks and controls the activities within each workstation with about a five minute planning horizon. A workstation may consist of a group of machines, such as one or more closely coupled machine tools, robots, inspection machines, materials transport devices, and part and tool buffers. Plans are developed and commands are issued to equipment to operate on material, tools, and fixtures in order to produce parts. The world model symbolic database contains names and attributes of parts, tools, and buffer trays in the workstation. Maps describe the location of parts, tools, and buffer trays.

Level 4—Equipment task

The equipment level schedules tasks and controls the activities of each machine within a workstation with about a 30 second planning horizon. (Tasks that take much longer may be

broken into several 30 second segments at the workstation level.) Level 4 decomposes each equipment task into elemental moves for the subsystems that make up each piece of equipment. Plans are developed that sequence elemental movements of tools and grippers, and commands are issued to move tools and grippers so as to approach, grasp, move, insert, cut, drill, mill, or measure parts. The world model symbolic database contains names and attributes of parts, such as their size and shape (dimensions and tolerances) and material characteristics (mass, color, hardness, etc.). Maps consist of drawings that illustrate part shape and the relative positions of part features.

Level 3—Elemental move (E-move)

The E-move level schedules and controls simple machine motions requiring a few seconds, such GO-ALONG-PATH, MOVE-TO-POINT, MILL-FACE, DRILL-HOLE, MEASURE-SURFACE, etc. (Motions that require significantly more time may be broken up at the task level into several elemental moves.) Plans are developed and commands are issued that define safe path waypoints for tools, manipulators, and inspection probes so as to avoid collisions and singularities, and assure part quality and process safety. The world model symbolic database contains names and attributes of part features such as surfaces, holes, pockets, grooves, threads, chamfers, burrs, etc. Maps consist of drawings that illustrate feature shape and the relative positions of feature boundaries.

Level 2—Primitive

The primitive level plans paths for tools, manipulators, and inspection probes so as to minimize time and optimize performance. It computes tool or gripper acceleration and deceleration profiles taking into consideration dynamical interaction between mass, stiffness, force, and time. Planning horizons are on the order of 300 milliseconds. The world model symbolic database contains names and attributes of linear features such as lines, trajectory segments, and vertices. Maps (when they exist) consist of perspective projections of linear features such as edges, lines or of tool or end-effector trajectories.

Level 1—Servo level

The servo level transforms commands from tool path to joint actuator coordinates. Planners interpolate between primitive trajectory points with a 30 millisecond look ahead. Executors servo individual actuators and motors to the interpolated trajectories. Position, velocity, or force servoing may be implemented, and in various combinations. Commands that define actuator torque or power are output every 3 milliseconds (or whatever rate is dictated by the machine dynamics and servo performance requirements). The servo level also controls the output drive signals to discrete actuators such as relays and solenoids. The world model symbolic database contains values of state variables such as joint positions, velocities, and forces, proximity sensor readings, position of discrete switches, condition of touch probes, as well as image attributes associated with camera pixels. Maps consist of camera images and displays of sensor readings.

At the Servo and Primitive levels, the command output rate is perfectly regular. At the E-Move level and above, the command output rates typically are irregular because they are event driven.

## Section 4. Organization and Timing in the RCS Hierarchy

Figure 4 summarizes the relationship in an RCS-4 system between the organizational hierarchy that is defined by the command tree, the computational hierarchy that is defined along each chain of command, and the behavioral hierarchy that is produced in state-space as a function of time. The behavioral hierarchy consists of state/time trajectories that are produced by computational modules executing tasks in real-time.

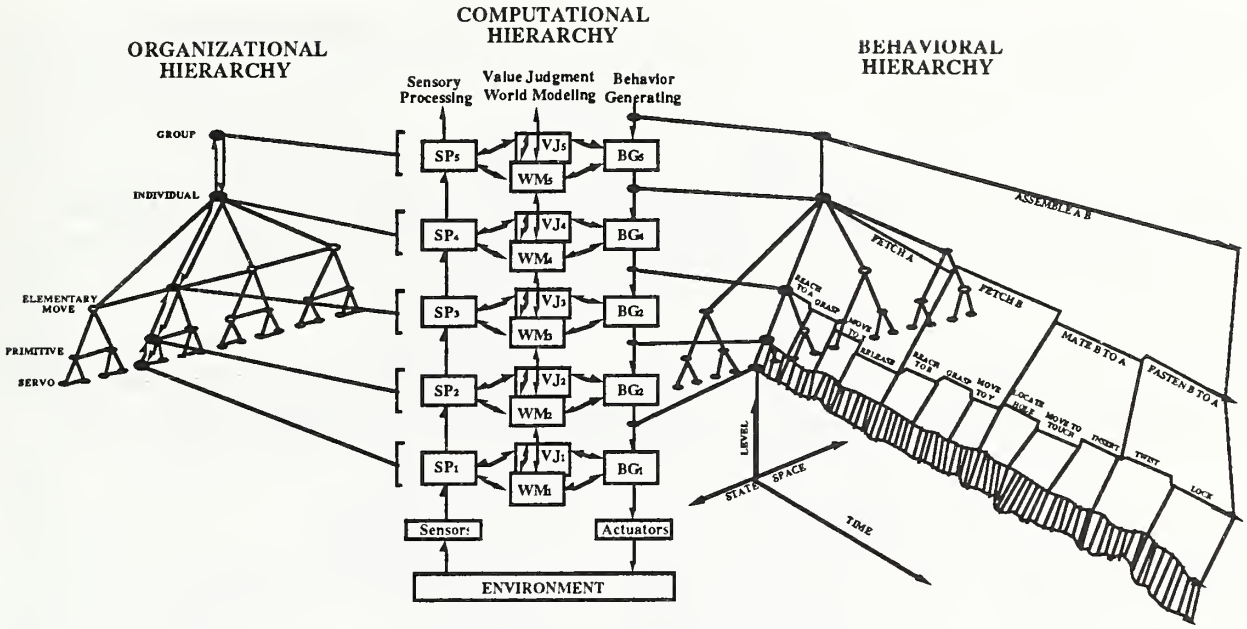



Figure 4. The relationship in RCS-4 between the organizational hierarchy of subsystems, the computational hierarchy of computing modules, and behavioral hierarchy of state trajectories that result as the computing modules execute tasks.

Levels in the RCS command hierarchy are defined by temporal and spatial decomposition of goals and tasks into levels of resolution, as well as by spatial and temporal integration of sensory data into levels of aggregation. Temporal resolution is manifested in terms of loop bandwidth, sampling rate, and state-change intervals. Temporal span is measured in length of historical traces and planning horizons. Spatial resolution is manifested in the resolution of maps and grouping of elements in subsystems. Spatial span is measured in range of maps and the span of control.

Figure 5 is a timing diagram that illustrates the temporal relationships in a RCS hierarchy containing seven levels of task decomposition and sensory processing. The sampling rate, the command update rate, the rate of subtask completion, and the rate of subgoal events, increases at the lower levels of the hierarchy, and decreases at upper levels of the hierarchy. The particular numbers shown in Figure 5 simply illustrate the relative timing between levels. Specific timing requirements are dependent on specific machines and applications.

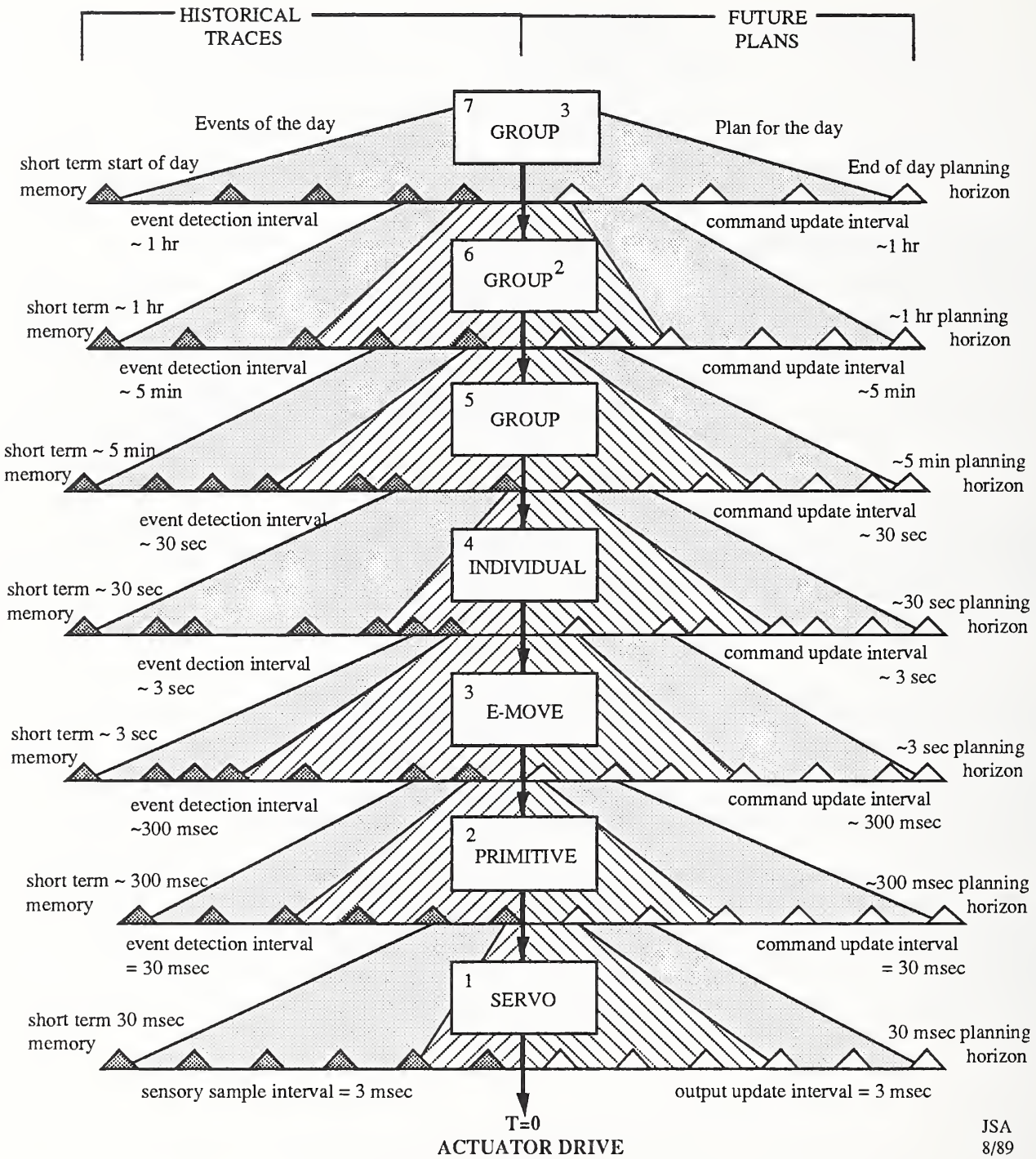



Figure 5. A timing diagram illustrating the temporal flow of activity in the task decomposition and sensory processing systems.

A fundamental principle of RCS-3 and 4 is that planning goes on simultaneously and continuously at all levels. At each level, planners periodically generate plans containing, on average, about ten steps. At each successive level, the first one or two steps in the current plan from the level above are further decomposed into subplans with an order of magnitude more resolution, and an order of magnitude less span in time and space. As a result, the average period between task commands decreases by about an order of magnitude at each lower level.

Replanning is done either at cyclic intervals, or whenever emergency conditions arise. The cyclic replanning interval may be as short as the average command update interval at each level, or about about ten percent of the planning horizon at each level. This allows the real-time planner to generate a new plan about as often as the executor generates a new output command. Less frequent replanning will reduce the computational speed requirements for real-time planning, but may also reduce control system effectiveness. Emergency replanning begins immediately upon the detection of an emergency condition.

Task execution also goes on simultaneously and continuously at all levels. At each level, executors periodically sample feedback, compare it with the current plan, and issue output commands to correct for deviations between plans and observations. At each sampling period feedback is tested for emergency conditions, and preplanned emergency recovery routines are invoked immediately upon detection of any emergency condition. This allows the system to react to emergencies within a single control cycle with preplanned recovery routines that bridge the temporal gap between when an emergency is detected and when the appropriate planners have new plans ready for execution.

As can be seen in Figure 5, there exists a duality between the task decomposition and the sensory processing hierarchies. At each hierarchical level a task can be decomposed into a set of subtasks at the next lower level. At each level a sensory event can be composed from a sequence of events at the next lower level, a recognized entity can be composed from a group of subentities at the next lower level. At each level, the sensory processing modules look back into the past about as far the planner modules look forward into the future. At each level, future plans have about the same detail as historical traces.

## Section 5. Task Decomposition

Each behavior generation (BG), or task decomposition (TD), module at each level consists of three sublevels: Job Assignment, Planning, and Execution.

The Job Assignment sublevel—JA submodule

The JA submodule is responsible for spatial task decomposition. It partitions the input task command into N spatially distinct jobs to be performed by N physically distinct subsystems, where N is the number of subsystems currently assigned to the TD module.

The JA submodule is also responsible for assigning tools and allocating physical resources (such as arms, hands, sensors, tools, and materials) to each of its subordinate subsystems for their use in performing their assigned jobs.

The Planner sublevel—PL(j) submodules, $j = 1, 2, \ldots, N$

For each of the N subsystems, there exists a planner submodule PL(j). Each planner submodule is responsible for decomposing the job assigned to its subsystem into a temporal sequence of planned subtasks. Each planner submodule is also responsible for resolving conflicts and mutual constraints between hypothesized plans of submodules.

The Executor sublevel—EX(j) submodules

There is an executor EX(j) for each planner PL(j). The executor submodules are responsible for successfully executing the plan state-graphs generated by their respective planners. When an executor is informed by the world model that a subtask in its current plan is successfully completed, the executor steps to the next subtask in that plan. When all the subtasks in the current plan are successfully executed, the executor steps to the first subtask in the next plan. If the

feedback indicates the failure of a planned subtask, the executor branches immediately to a preplanned emergency subtask. Its planner simultaneously begins work selecting or generating an error recovery sequence which can be substituted for the former plan which failed. Output subcommands produced by executors at level i become input commands to job assignment submodules in TD modules at level i-1.

Planners PL(j) constantly operate in the future, each generating a plan to the end of its planning horizon. The executors EX(j) always operate in the present, at time t=0, constantly monitoring the current state of the world reported by feedback from the world model.

At each level, each executor submodule closes a reflex arc, or servo loop, and the executor submodules at the various hierarchical levels form a set of nested servo loops. The executor loop bandwidth decreases about an order of magnitude at each higher level. Each level has a dominant frequency of execution. The actual frequency is dictated by the physical equipment and the application.

## Task Knowledge

Fundamental to task decomposition is the representation and use of task knowledge. A task is a piece of work to be done, or an activity to be performed. For any TD or BG module, there exists a set of tasks that that module knows how to do. Each task in this set can be assigned a name. The task vocabulary is the set of task names assigned to the set of tasks each TD or BG module is capable of performing.

Knowledge of how to perform a task may be represented in a frame data structure. An example task frame is:

| | |
|---|---|
| TASKNAME | Name of the task |
| Goal | Event or condition that successfully terminates the task |
| Object | Identification of thing to be acted upon |
| Parameters | Priority<br>Status (e.g. active, waiting, inactive)<br>Timing (e.g. speed, completion time)<br>Coordinate system in which task is expressed<br>Stiffness matrices<br>Tolerances |
| Agents | Identification of subsystems that will perform the task |
| Requirements | Feedback information required from the world model during the task<br>Tools, time, resources, and materials needed to perform the task<br>Enabling conditions that must be satisfied to begin or continue the task<br>Disabling conditions that will interrupt or abort the task activity |
| Procedures | Pre-computed plans or scripts for executing the task<br>Planning algorithms<br>Functions that may be called<br>Emergency procedures for each disabling condition |

Figure 6. A typical task frame

The name of the task is a string that identifies the type of activity to be performed. The

goal may be a vector that defines an attractor value, set point, or desired state to be achieved by the task. The goal may also be a map, graph, or geometric data structure that defines a desired "to-be" condition of an object, or arrangement of components.

The requirements section includes information required during the task. This may consist of a list of state variables, maps, and/or geometrical data structures that convey actual, or "as-is" conditions that currently exist in the world. Requirements may also include resources, tools, materials, time, and conditions needed for performing the task.

The procedure section contains either a set of pre-computed plans or scripts for decomposing the task, or one or more planning algorithms for generating a plan, or both. For example, the procedure section may contain a set of IF/THEN rules that select a plan appropriate to the "as-is" conditions reported by the world model. Alternatively, the procedure section may contain a planning algorithm that computes the difference between "to-be" and "as-is" conditions. This difference may then be treated as an error that the task planner attempts to reduce, or null through "Means/Ends Analysis" or A* search. Each subsystem planner would then develop a sequence of subtasks designed to minimize its subsystem error over an interval from the present to its planning horizon. In either case, each executor would act as a feedback controller, attempting to servo its respective subsystem to follow its plan. The procedure section also contains emergency procedures that can be executed immediately upon the detection of a disabling condition.

In plans involving concurrent job activity by different subsystems, there may be mutual constraints. For example, a start-event for a subtask activity in one subsystem may depend on the goal-event for a subtask activity in another subsystem. Some tasks may require concurrent and cooperative action by several subsystems. This requires that both planning and execution of subsystem plans be coordinated. (The reader should not infer from this discussion or others throughout this paper, that all these difficult problems have been solved, at least not for the general case. Much remains unknown that will require extensive further research. The RCS architecture simply provides a framework wherein each of these problems can be explicitly represented and input/output interfaces can be defined. In several RCS designs, human operators are an integral part of some computational nodes, so that tasks that cannot yet be done automatically are done interactively by humans. In the NIST Automated Deburring and Chamfering System workstation, for example, the tasks of the fixturing subsystem are performed by a human operator.)

The library of task frames that reside in each TD module define the capability of the TD module. The names of the task frames in the library define the set of task commands that TD module will accept. There, of course, may be several alternative ways that a task can be accomplished. Alternative task or job decompositions can be represented by an AND/OR graph in the procedure section of the task frame.

The agents, requirements, and procedures in the task frame specify for the TD module "how to do" commanded tasks. This information is a-priori resident in the task frame library of the TD module. The goal, object, and parameters specify "what to do", "on what object", "when", "how fast", etc. Figure 7 illustrates the instantiation of a task frame residing in the TD module library by a task command. When a TD module inputs a task command, it searches its library of task frames to find a task name that matches the command name. Once a match is found, the goal, object, and parameter attributes from the command are transferred into the task frame. This activates the task frame, and as soon as the requirements listed in the task frame are met, the TD module can begin executing the task plan that carries out the job of task decomposition.

Task knowledge is typically difficult to discover, but once known, can be readily used and duplicated. For example, the proper way to mill a pocket, drill a hole, or fixture a part may be difficult to derive from first principles. However, once such knowledge is known and represented in a task frame, it is relatively easy to transform into executable code.

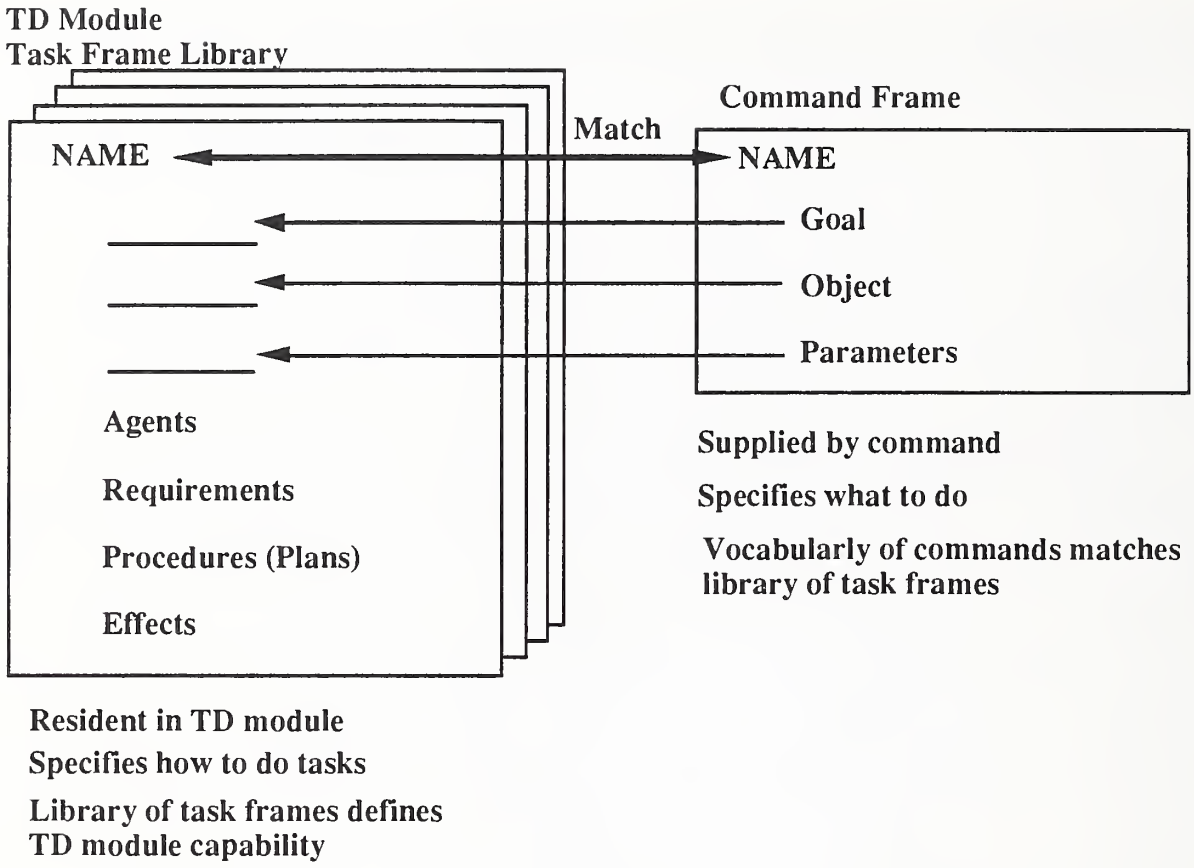



Figure 7. The relationship between commands and task frames.

## Section 5. World Modeling

The world model is an intelligent system's internal representation of the external world. It is the system's best estimate of objective reality. It acts as a buffer between sensory processing and task decomposition. This enables the task decomposition system to act on information that may not be immediately or directly observable by sensors, and enables the sensory processing system to do recursive estimation based on a number of past sensory measurements. The world model is hierarchically structured and distributed such that there is a world model (WM) module with Knowledge Database (KD) in each node at every level of the RCS control hierarchy. The KD forms a passive, hierarchically structured, distributed data store. The WM provides the operations that act upon the KD. At each level, the WM and value judgment (VJ) modules perform the functions illustrated in Figure 8 and described below.

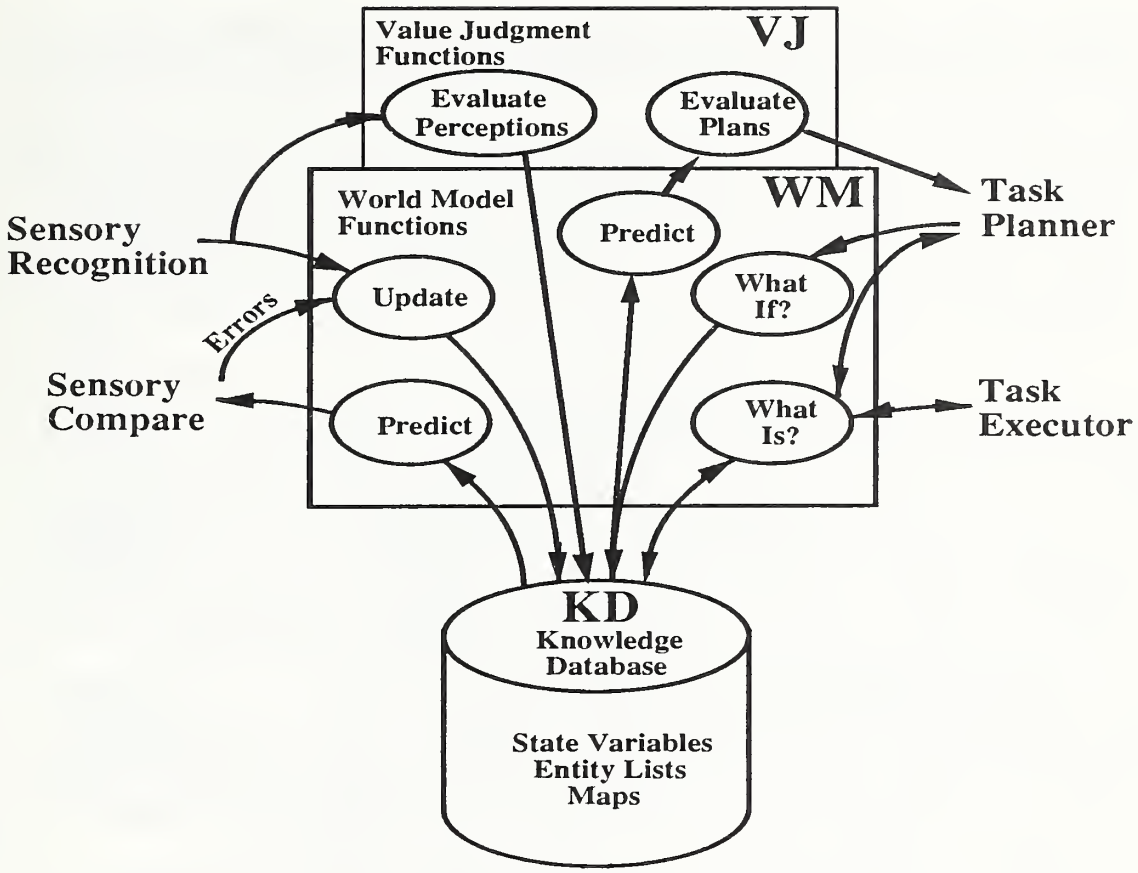



Figure 8. Functions performed by the WM module. 1) Update the knowledge database with recognized entities. 2) Predict sensory data. 3) Answer "What is?" queries from task executor and return current state of world. 4) Answer"What if?" queries from task planner and predict results for evaluation.

1) WM modules maintain the KD knowledge database, keeping it current and consistent. In this role, the WM modules perform the functions of a database management system. They update KD state estimates based on correlations and differences between world model predictions and sensory observations at each hierarchical level. The WM modules enter newly recognized entities, states, and events into the KD database, and delete entities and states determined by the sensory processing modules to no longer exist in the external world. The WM modules also enter estimates, generated by the VJ modules, of the reliability of KD state variables. Believability or confidence factors are assigned to many types of state variables.

2) WM modules generate predictions of expected sensory input for use by the appropriate sensory processing SP modules. In this role, a WM module performs the functions of a graphics engine, or state predictor, generating predictions that enable the sensory processing system to perform correlation and predictive filtering. WM predictions are based on the state of the task and estimated states of the external world. For example in vision, a WM module may use the information in an object frame to generate predicted images which can be compared pixel by pixel, or entity by entity, with observed images.

3) WM modules answer "What is?" questions asked by the planners and executors in corresponding BG modules. In this role, the WM modules perform the function of database

query processors, question answering systems, or data servers. World model estimates of the current state of the world are used by BG or TD module planners as a starting point for planning. Current state estimates are also used by BG or TD module executors for servoing and branching on conditions.

4) WM modules answer "What if?" questions asked by the planners in the corresponding level BG or TD modules. In this role, the WM modules perform the function of simulation by generating expected status resulting from actions hypothesized by the planners. Results predicted by WM simulations are sent to value judgment VJ modules for evaluation. For each BG or TD hypothesized action, a WM prediction is generated, and a VJ evaluation is returned to the BG or TD planner. This BG-WM-VJ loop enables BG planners to select the sequence of hypothesized actions producing the best evaluation as the plan to be executed.

## The Knowledge Database

The world model knowledge database (KD) includes both a-priori information which is available to the intelligent system before action begins, and a-posterior knowledge which is gained from sensing the environment as action proceeds. The KD represents information about space, time, entities, events, states of the world, and laws of nature. Knowledge about space is represented in maps. Knowledge about entities, events, and states is represented in lists and frames. Knowledge about the laws of physics, chemistry, optics, and the rules of logic and mathematics is represented in the WM functions that generate predictions and simulate results of hypothetical actions. Laws of nature may be represented as formulae, or as IF/THEN rules of what happens under certain situations, such as when things are pushed, thrown, or dropped.

The world model also includes knowledge about the intelligent system itself, such as the values assigned to goal priorities, attribute values assigned to objects, and events; parameters defining kinematic and dynamic models of robot arms or machine tool stages; state variables describing internal pressure, temperature, clocks, fuel levels, body fluid chemistry; the state of all the currently executing processes in each of the TD, SP, WM, and VJ modules; etc.

The correctness and consistency of world model knowledge is verified by sensors and sensory processing SP mechanisms that measure differences between world model predictions and sensory observations. These differences may be used by recursive estimation algorithms to keep the world model state variables the best estimates of the state of the world. Attention algorithms may be used to limit the number of state variables that must be kept up-to-date and any one time.

Information in the world model knowledge database may be organized as state variables, system parameters, maps, and entity frames.

### State variables

State variables define the current value of entity and system attributes. A state variable may define the state of a clock, the position, orientation, and velocity of a gripper, the position, velocity, and torque of an actuator, or the state of a computing module.

### System parameters

System parameters define the kinematic and dynamic characteristics of the system being controlled, e.g., the inertia of objects, machines, and tools. System parameters may also define coordinate transforms necessary to transform commands and sensory information from one working coordinate system to another.

### Entity Frames

An entity frame is a symbolic list structure, in which the ENTITY NAME is the list head, and in which knowledge about the entity is stored as attribute-value pairs (or attribute-list pairs). The world model contains a list of all the entities that the intelligent system knows about. A subset of this list is the set of current entities known to be present in any given situation. A subset of the list of current entities is the set of entities of attention. These are the entities that the system is currently acting upon, or is planning to act upon momentarily.

There are two types of entities: generic and specific. A generic entity is an example of a

class of entities. A generic entity frame contains the attributes of its class. A specific entity is a particular instance of an entity. A specific entity frame inherits the attributes of its class.

Different levels of entities exist at different levels of the hierarchy. At level 1, entity frames describe points; at level 2, entity frames describe lines and vertices; at level 3, they describe surfaces; at level four, objects; and at level 5, groups; at level 6 and above, entity frames describe higher order groups.

An example of an entity frame is shown in Figure 9.

| | |
|---|---|
| **ENTITY NAME** | -- part id#, lot#, etc. |
| kind | -- model# |
| type | -- generic or specific |
| level | -- point, line, surface, object, group |
| position | -- map location of center of mass (time, uncertainty) |
| orientation | -- coordinate axes directions (time, uncertainty) |
| coordinates | -- coordinate system of map |
| dynamics | -- velocity,acceleration (time, uncertainty) |
| trajectory | -- sequence of positions (time, uncertainty) |
| geometry | -- center of gravity (uncertainty)<br>axis of symmetry (uncertainty)<br>size (uncertainty)<br>boundaries (uncertainty) |
| subentities | -- pointers to lower level entities that make up named entity |
| parent entity | -- pointer to higher level entity of which named entity is part |
| properties | -- mass, color, hardness, smoothness, etc. |
| value | -- sunk cost, value at completion |
| due date | -- date required |

Figure 9. An example entity frame

## Maps

Maps describe the distribution of entities in space. Each point, or pixel, on a map may have a pointer that points to the name of the entity that projects to that point on the map. A pixel may also have one or more attribute-value pairs. For example, a map pixel may have a brightness, or color (as in an image), or an altitude (as in a topographic map). A map may also be represented by a graph that indicates routes between locations, for example, the routes available to robot carts moving between workstations.

Any specific map is defined in a particular coordinate frame. There are three general types of map coordinate frames that are important: world coordinates, object coordinates, and egospheres. An egosphere is a spherical coordinate system with the intelligent system at the origin and properties of the world are projected onto the surface of the sphere. World, object, and egosphere coordinate frames are discussed more extensively in [1].

## Map-Entity Relationships

Map and entity representations may be cross referenced in the world model as shown in Figure 10. For example, each entity frame may contain a set of geometrical and state parameters that enables the world model to project that entity onto a map. The world model can thus compute the set of egosphere or world map pixels covered by an entity. By this means, entity parameters can be inherited by map pixels, and hence entity attributes can be overlaid on maps.

Conversely, each pixel on a map may have pointers to entities covered by that pixel. For example, a pixel may have a pointer to a point entity whose frame contains the projected distance or range to the point covered by that pixel. Each pixel may also have a pointer to a line entity frame

indicating the position and orientation of an edge, line, or vertex covered by the pixel. Each pixel may also have a pointer to a surface entity indicating position and orientation of a surface covered by the pixel. Each pixel may have a pointer to an object entity indicating the name of the object covered, and a pointer to a group entity indicating the name of the group covered.

The world model thus provides cross referencing between pixel maps and entity frames. Each level of world modeling can thus predict what objects will look like to sensors, and each level of sensory processing can compare sensory observations with world model predictions. (The reader should not infer from this discussion that such a cross-coupled systems have been fully implemented, or that it is even well understood how to implement such systems for real-time operations. What is described is the kind of cross-coupled system that will be necessary in order to achieve truly intelligent robotic systems. It seems likely that special purpose hardware and firmware will need to be developed. Clearly, much additional research remains to be done before these problems are solved. The RCS reference model architecture simply defines how such processes should be organized and what interfaces need to be defined.

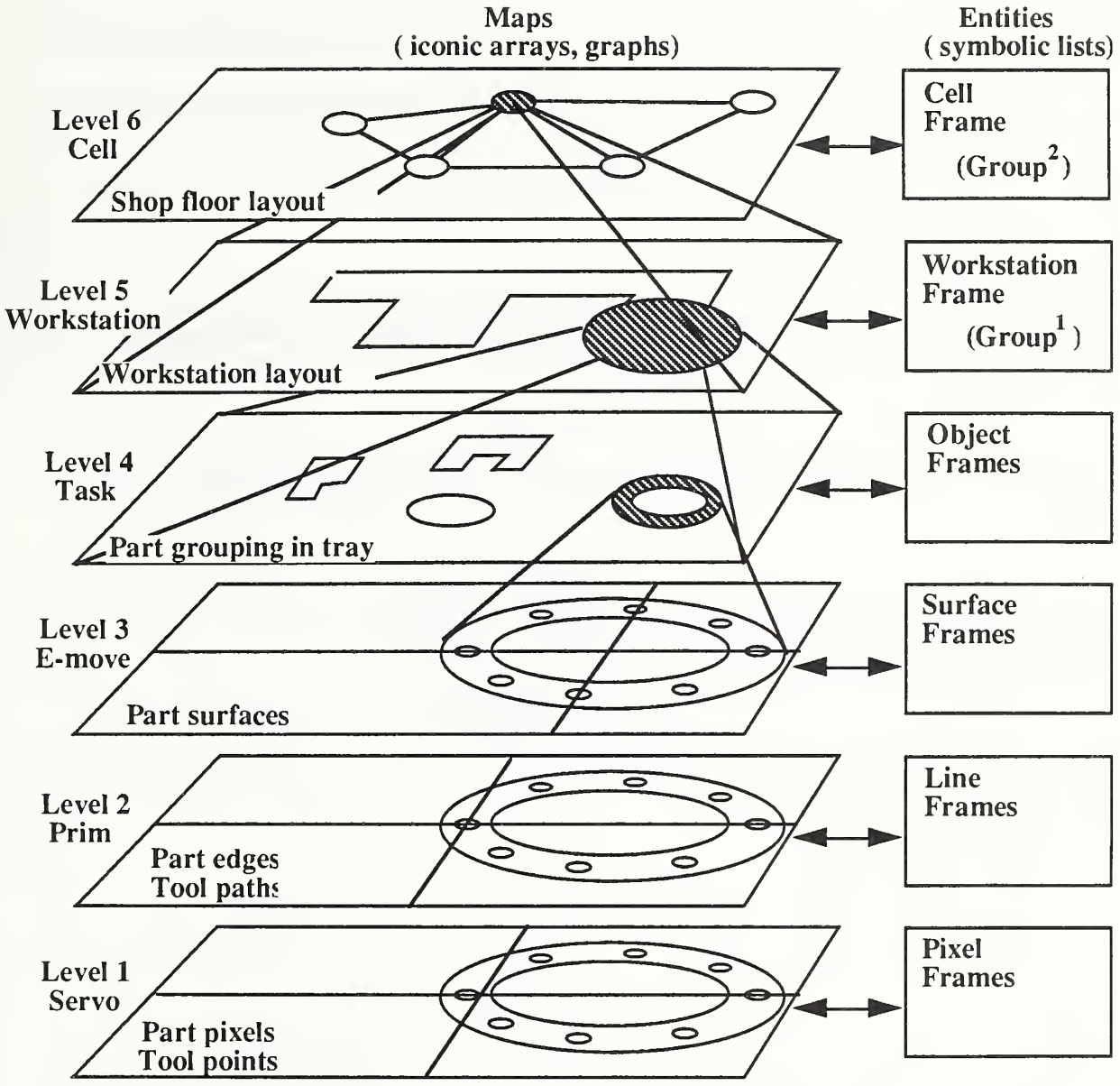



Figure 10. Map-entity relationships in a world model for manufacturing. The world model provides processes by which symbolic entity frames can be transformed into maps, and vice versa.

An example of a knowledge database for an intelligent robot in a manufacturing workstation such as shown in Figure 3 might be the following:

Level 1

State Variables

State clock and sync signals
State vector that defines the best estimate of the position, velocity, and force of each joint actuator.
Estimated time or distance to nearest point entity contact

System parameters

Joint limit margin for each actuator
Gravity compensation matrix
Inertia matrix
Forward and inverse kinematic transform from end-effector to joint coordinates
Forward and inverse transform from sensor egosphere to end-effector egosphere

Entity frames

Point entity frames for entities of attention
Pixel frames for sensor egosphere map pixels

Maps

Sensor egosphere map overlaid with projections of point entities of attention

Level 2

State variables

State clock and sync signals
State vector defining the best estimate of load or tool pose, velocity, force, etc.
Estimated time or distance to nearest linear entity contact
Singularity clearance margins

System parameters

Forward and inverse force and velocity coordinate transform from equipment to end-effector coordinates
Forward and inverse transform from end-effector egosphere to equipment centered inertial egosphere coordinates
Manipulator dynamic model
Load dynamic model
Limits of travel in coordinate system of command
Positions of singularities
Position, velocity, and force limits

Entity frames

Linear entity frames (edges, lines, trajectories, vertices, etc.) for entities of attention

Maps

Tool or end-effector egosphere map overlaid with projection of linear entities of attention and their trajectories (observed and planned)

Level 3

State variables

State clock and sync signals
Best fit trajectories for observed poses, velocities, and forces of load or tool
Estimated time or distance to nearest surface entity contact
Estimated minimum clearance for singularities and obstacle surfaces

System parameters

Forward and inverse transform from equipment centered inertial egosphere coordinates to part coordinates
Manipulator geometry and dynamic model
Load geometry and dynamic model
Limits of travel in coordinate system of command

Position, velocity, and force limits
Cost/benefit function parameters for analysis of hypothesized path plans

Entity frames

Surface entity frames for entities of attention

Maps

Equipment centered inertial egosphere map overlaid with projection of surface entities of attention and their swept volumes (observed and planned)

Level 4

State variables

State clock and sync signals from other equipment
Best estimate observed degree of task completion
State of task enabling and disabling conditions
Predicted results of plan

System parameters

Task enabling and disabling conditions
Forward and inverse transform from equipment centered inertial egosphere to equipment centered world coordinates
Equipment geometry and dynamic model
Part and tool geometry and dynamic model
Limits of travel in coordinate system of command
Cost/benefit function for evaluation of plan results

Entity frames

Object entity frames for objects of attention (trays, fixtures, tool racks, free space, parts, grippers, tools, fasteners, etc.)

Maps

Equipment centered world map overlaid with projections of objects of attention; also overlaid with projections of planned sequence of E-moves

Level 5

State variables

State clock and sync signals from other equipment
Observed degree of task completion
State of task enabling and disabling conditions
Predicted results of hypothesized plan

System parameters

Forward and inverse transforms from workstation to equipment centered world coordinates
Task enabling and disabling conditions
Workstation task timing model
Cost/benefit evaluation function for hypothesized plan results

Entity frames

Workstation equipment entity frames
Group entity frames (objects grouped in trays, buffers, tool racks, fixtures, etc.) for groups of attention

Maps

Workstation centered world map overlaid with projections of equipment entities and group entities of attention; also overlaid with planned sequence of robot task

Additional discussion of RCS world models for robots can be found in [16].

## Section 6. Sensory Processing

Sensory processing is the mechanism of perception. Perception is the establishment and maintenance of correspondence between the internal world model and the external real world. The function of sensory processing is to extract information about entities, events, states, and relationships in the external world, so as keep the world model accurate and up to date.

### Map Updates

World model maps may be updated by sensory measurement of points, edges, and surfaces. Such information may be derived from vision, touch, sonar, radar, or laser sensors. The most direct method of measuring points, edges, and surfaces is through touch. In the manufacturing environment, touch probes are used by coordinate measuring machines to measure the 3-D position of points. Touch probes on machine tools and tactile sensors on robots can be used for making similar measurements. From such data, sensory processing algorithms can compute the orientation and position of surfaces, the shape of holes, the distance between surfaces, and the dimensions of parts.

Other methods for measuring points, edges, and surfaces include stereo vision, photogrammetry, laser ranging, structured light, image flow, acoustic ranging, and focus-based optical probes. Additional information about surface position and orientation may also be computed from shading, shadows, and texture gradients. Each of these various methods produce different accuracies and have differing computational and operational requirements.

### Recognition and Detection

Recognition is the establishment of a one-to-one match, or correspondence, between a real world entity and a world model entity . The process of recognition may proceed top-down, or bottom-up, or both simultaneously. For each entity in the world model, there exists a frame filled with information that can be used to predict attributes of corresponding entities observed in the world. The top-down process of recognition begins by hypothesizing a world model entity and comparing its predicted attributes with those of the observed entity. When the similarities and differences between predictions from the world model and observations from sensory processing are integrated over a space-time window that covers an entity, a matching, or cross-correlation value is computed between the entity and the model. If the correlation value rises above a selected threshold, the entity is said to be recognized. If not, the hypothesized entity is rejected and another tried.

The bottom-up process of recognition consists of applying filters and masks to incoming sensory data, and computing image properties and attributes. These may then be stored in the world model, or compared with the properties and attributes of entities already in the world model. Both top-down and bottom-up processes proceed until a match is found, or the list of world model entities is exhausted. Many perceptual matching processes may operate in parallel at multiple hierarchical levels simultaneously.

If a SP module recognizes a specific entity, the WM at that level updates the attributes in the frame of that specific WM entity with information from the sensory system.

If the SP module fails to recognize a specific entity, but instead achieves a match between the sensory input and a generic world model entity, a new specific WM entity will be created with a frame that initially inherits the features of the generic entity. Slots in the specific entity frame can then be updated with information from the sensory input.

If the SP module fails to recognize either a specific or a generic entity, the WM may create an "unidentified" entity with an empty frame. This may then be filled with information gathered from the sensory input.

When an unidentified entity occurs in the world model, the behavior generation system may (depending on other priorities) select a new goal to <identify the unidentified entity>. This may initiate an exploration task that positions and focuses the sensor systems on the unidentified entity, and possibly even probes and manipulates it, until a world model frame is constructed that adequately describes the entity. The sophistication and complexity of the exploration task depends on task knowledge about exploring things. Such knowledge may be very advanced and include

sophisticated tools and procedures, or very primitive. Entities may, of course, simply remain labeled as "unidentified," or "unexplained."

Detection of events is analogous to recognition of entities. Observed states of the real world are compared with states predicted by the world model. Similarities and differences are integrated over an event space-time window, and a matching, or cross-correlation value is computed between the observed event and the model event. When the cross-correlation value rises above a given threshold, the event is detected.

## Sensory Processing Modules

The Sensory Processing (SP) modules are responsible for gathering data from sensors, filtering and integrating this data, and interpreting it. Noise rejection techniques such as Kalman filtering are implemented here, as well as feature extraction, pattern recognition, and image understanding. At the upper levels of the hierarchy, more abstract interpretations of sensory data are performed and inputs from a variety of sensor systems are integrated over space and time into a single interpretation of the the external world.

Each SP module at each level consists of five types of operations: 1) Coordinate transformation, 2) Comparison, 3) Temporal Integration, 4) Spatial Integration, and 5) Recognition/Detection, as shown in Figure 11.

### 1) Coordinate transformation

Before a world model estimated variable can be compared with an observed sensory variable, the estimated and observed variables must be registered with each other in the same coordinate system. Sensory variables are most often represented in sensor egosphere coordinates. World model estimates may be represented in a variety of coordinate frames, each of which have certain computational advantages. Among these are head egosphere, inertial egosphere, part or tool egosphere, or world coordinates with origins at convenient points.

### 2) Comparison

Each comparator matches an observed sensory variable (such as an image pixel attribute) with a world model prediction of that variable. The predicted variable may be subtracted from the observed, or multiplied by it, to obtain a measure of similarity (correlation) and difference (variance) between the observed variable and the predicted variable. Similarities indicate the degree to which the WM predictions are correct, and hence are a measure of the correspondence between the world model and reality. Differences indicate a lack of correspondence between world model predictions and sensory observations. Differences imply that either the sensor data or world model is incorrect. Difference images from the comparator go three places:

a) They are returned directly to the WM for real-time local pixel attribute updates. This produces a tight feedback loop whereby the world model predicted image becomes an array of recursive state-estimations. Difference images are error signals used to make each pixel attribute in the predicted image a "best estimate" of the corresponding pixel attribute in the current sensory input.

b) Difference images are also transmitted upward to spatial and temporal integrators where they are integrated over time and space in order to recognize and detect entity attributes. This integration constitutes a summation, correlation, or "chunking", of sensory data into entities. At each level, lower order entities are "chunked" into higher order entities, i.e. points are chunked into lines, lines into surfaces, surfaces into objects, objects into groups, etc.

c) Difference images also are transmitted to the VJ module at the same level where statistical parameters are computed so as to assign uncertainty and believability factors to estimates of pixel entity attributes.

### 3) Temporal integration

Temporal integration accumulates similarities and differences between predictions and observations over intervals of time. Temporal integration operating just on sensory data can produce a summary, such as a total, or average, of sensory information over the given time window. Temporal integrators operating on the similarity and difference values computed by comparators may produce temporal cross-correlation and covariance functions between the model

and the observed data. These correlation and covariance functions are measures of how well the dynamic properties of the world model entity match those of the real world entity. The boundaries of the temporal integration window may be derived from world model prediction of event durations, or from behavior generation parameters such as sensor fixation periods.

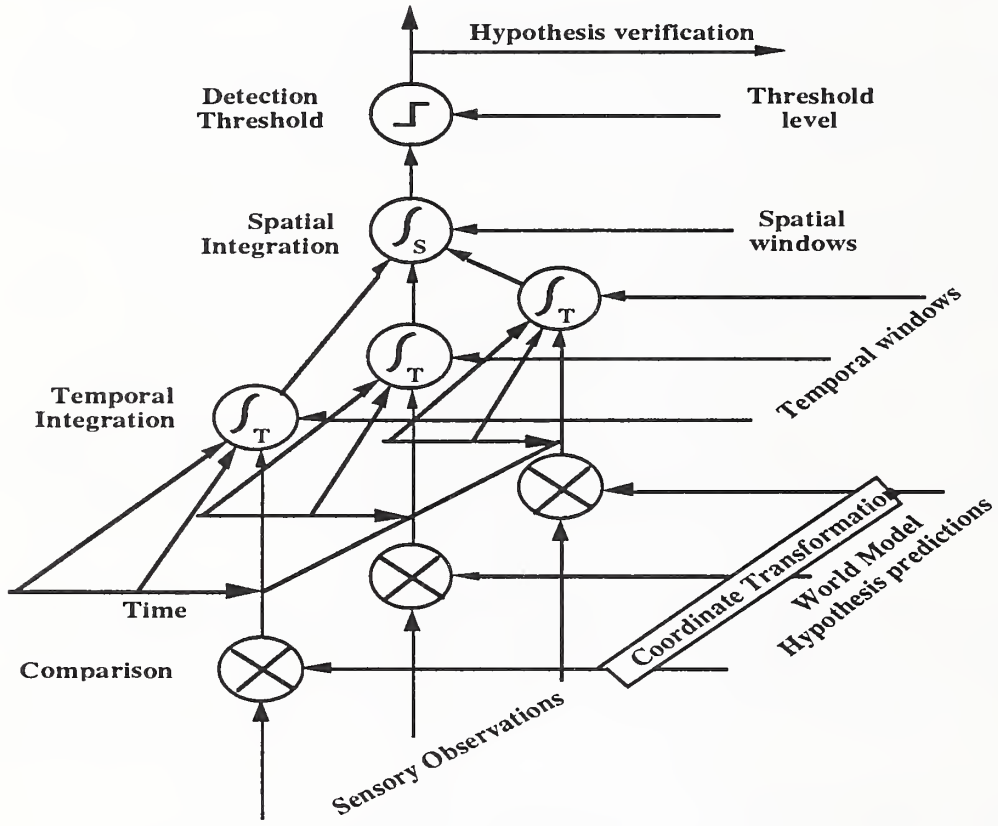



Figure 11. Each sensory processing SP module consists of: 1) a set of coordinate transformers, 2) a set of comparators that compare sensory observations with world model predictions, 3) a set of temporal integrators that integrate similarities and differences, 4) a set of spatial integrators that fuse information from different sensory data streams, and 5) a set of threshold detectors that recognize entities and detect events

4) Spatial integration

Spatial integration accumulates similarities and differences between predictions and observations over regions of space. This produces spatial cross-correlation or convolution functions between the model and the observed data. Spatial integration summarizes sensory information from multiple sources at a single point in time. It determines whether the geometric properties of a world model entity match those of a real world entity. For example, the product of an edge operator and an input image may be integrated over the area of the operator to obtain the correlation between the image and the edge operator over the region covered by the filter. The limits of the spatial integration window may be determined by world model predictions of entity size and shape. In some cases, the order of temporal and spatial integration may be reversed, or interleaved.

5) Recognition/Detection threshold

When the spatio-temporal correlation function exceeds some threshold, object recognition (or event detection) occurs. For example, if the spatio-temporal summation over the area of an edge operator exceeds threshold, an edge is said to be recognized at the center of the area.

### The Sensory Processing Hierarchy

It has long been known that sensory processing occurs in a hierarchy of processing modules, and that perception proceeds by "chunking," i.e. by recognizing patterns, groups, strings, or clusters of points at one level as a single feature, or point in a higher level, more abstract space. It also has been observed that this chunking process proceeds by about an order of magnitude per level, both spatially and temporally [17,18]. Thus, at each level in the proposed architecture, SP modules integrate, or chunk, information over space and time by about an order of magnitude.

In order to facilitate the comparison of predicted and observed variables, there must be a correspondence between the form of the sensory data being processed at each level of the sensory processing hierarchy and the form of the information stored in the world model knowledge base at that level. This is illustrated in Figure 10 where:

Level 1 of the sensory processing hierarchy compares point measurements with projected point entities from the world model. Differences are used to correct the estimated point entities. Similarities between observed and predicted point entities are integrated into line entities.

Level 2 of the sensory processing hierarchy compares observed line entities with projected line entities from the world model. Differences are used to correct the estimated line entities, and similarities are integrated into surface entities.

Level 3 compares observed surfaces with predicted surfaces. Differences are used to correct the predictions, and similarities are integrated to recognize objects.

Level 4 compares observed objects with predicted objects. Differences are used to correct the world model, and similarities are integrated into groups.

Level 5 compares observed and predicted group characteristics, updates the world model, and integrates information into larger groups. And so on.

Further discussion of the sensory processing hierarchy appears in [1].

## Section 7. Value Judgments

Value judgments provide the criteria for making intelligent choices. Value judgments evaluate the costs, risks, and benefits of plans and actions, and the desirability, attractiveness, and uncertainty of objects and events. Value judgment modules produce evaluations that can be represented as value state-variables. These can be assigned to the attribute lists in entity frames of objects, persons, events, situations, and regions of space. They can be assigned to map pixels and can become map overlays. Value state-variables can thus label objects, situations, or places as good or bad, friendly or hostile, attractive or repulsive.

Value judgment algorithms may evaluate risk and compute the level of uncertainty in the recognition of entities and the detection of events. Knowledge in the world model can thus be labeled as reliable or uncertain.

Value judgments can evaluate events as important or trivial. The utilization of memory can be optimized by storing only those events and situations evaluated as "important". Those events labeled as "trivial" can safely be forgotten and cleared from memory. Intelligent machines that include capabilities for learning require value judgment functions to indicate what to "reward" and what to "punish."

Cost/benefit value judgment algorithms can be used to evaluate plans or steer task execution so as to minimize cost and maximize benefits. Value state-variables can be assigned to the attribute lists of plans and actions in task frames to label tasks and plans as good or bad, costly or inexpensive, risky or safe, with high or low expected payoff. Priorities are value state-variables

that provide estimates of importance. Priorities can be assigned to task frames so that planners and executors can decide what to do first, how much effort to spend, how much risk is prudent, and how much cost is acceptable, for each task. Value state-variables can thus be used by the behavior generation modules both for planning and executing actions. They provide criteria for making decisions about which coarse of action to take. Priorities may also determine the degree of alertness, or tactics chosen for planning tasks in a hostile environment. A priority on safety may produce conservative tactics. A priority on aggression may produce an attack on an enemy. Such priorities are typically input from a human commander.

Intelligent systems designed for military purposes typically require value judgments labeling objects in the world model as "friend" or "foe." If an object is labeled "friend" it should be defended or assisted. If it is labeled "foe" it should be attacked or avoided. The computation of "friend or foe" may be accomplished by signal analysis or pattern recognition. This computation may be very difficult, or trivially simple. For example, in current battlefield situations, friendly systems typically carry transponders that actively emit signals, or modulate reflected energy in a distinctive way that makes recognition of "friend" easy, even in smoke or fog or at night. Often, anything else is assumed to be "foe".

### VJ Modules

Value state-variables may be assigned by a human programmer or operator, or they may be computed by value judgment functions residing in VJ modules. Inputs to VJ modules describe entities, events, situations, and states. Inputs may also include sensor signal-to-noise ratio, variance between sensory observations and world model predictions, degree of success in goal achievement, and reward or punishment signals from a variety of sources. VJ value judgment functions may compute measures of cost, risk, and benefit. VJ outputs are value state-variables that may be assigned to objects, events, or plans.

The VJ value judgment mechanism can typically be defined as a mathematical or logical function of the form

$$\mathbf{E} = \mathbf{V}(\mathbf{S})$$

where **E** is an output vector of value state-variables
**V** is a value judgment function that computes **E** given **S**
**S** is an input state vector defining conditions in the world model, including the self.

The components of **S** are entity attributes describing states of tasks, objects, events, or regions of space. These may be derived either from processed sensory information, or from the world model. The value judgment function **V** in the VJ module computes a numerical scaler value (i.e. an evaluation) for each component of **E** as a function of the input state vector **S**. The components of **E** may be assigned to attributes in the world model frame of various entities, events, or states. **E** is a time dependent vector.

Further discussion of value judgments can be found in [1].

## Conclusion

RCS is a reference model architecture that defines the types of functions that are required in a real-time intelligent control system, and how these functions are related to each other. RCS is not a system design, nor is it a specification of how to implement specific systems. It has, nevertheless, been found useful by many researchers and engineers for designing intelligent machine systems.

Systems based on the RCS architecture have been designed and more or less implemented for a wide variety of applications that include loading and unloading of parts and tools in machine tools, controlling machining workstations, performing robotic deburring and chamfering, and controlling space station telerobots, multiple autonomous undersea vehicles, unmanned land vehicles, coal mining automation systems, postal service mail handling systems, and submarine

operational automation systems. Software developers accustomed to using RCS for building control systems have found it provides a structured, even recursive, approach to design that makes it possible to reuse a great deal of software, not only within a single system, but between systems designed for significantly different applications.

Critics of RCS have objected to its rigidity, claiming "it forces a system to be built in a certain structure, even if it can be built more effectively in a different architecture". This, however, misses the point of a reference model architecture. A reference model architecture is a canonical form, not a system design specification. Reference models can be implemented in a variety of ways. For example, RCS could be implemented with neural nets (at least in principle). Many of the lower level features can be implemented by finite-state-machines and motion controllers. Higher level functions can be implemented by expert systems, lisp machines, transputers, connection machines, or special purpose processing engines. The RCS reference model architecture combines real-time motion planning and control with high level task planning, problem solving, world modeling, recursive state estimation, tactile and visual image processing, and acoustic signature analysis. In fact, the evolution of the RCS concept has been driven by an effort to include the best properties and capabilities of most, if not all, the intelligent control systems currently known in the literature, from subsumption to SOAR [19], from blackboards [20] to object-oriented programming [21].

However, the apparent completeness of the canonical form of RCS should not be taken to mean that all of the problems are solved in each of these areas, and all the issues are resolved. Far from it. The enormous complexity of the RCS architecture should lead one to appreciate how difficult the task of creating intelligent systems is going to be. Not so long ago, respected people were predicting that intelligent robots would be able to duplicate human performance before the end of this century. An correct understanding of the RCS reference model should lead one to realize that the creation of intelligence may be more complex and scientifically challenging than controlling fusion or analyzing the genome, and possibly more rewarding as well.

Current research efforts at NIST are directed towards understanding how to use the RCS reference model as a design tool. An RCS design paradigm begins with analysis of tasks and goals, then develops control algorithms, data structures, and sensory information necessary to perform the tasks and accomplish the goals. Attempts are underway to develop this approach into a formal methodology for engineering real-time intelligent control systems and analyzing their performance.

## Acknowledgements

The work described in this report was funded by National Institute of Standards and Technology, the U.S. Air Force Next Generation Controller project, the DARPA Advanced Submarine Technology project, and the U.S. Bureau of Mines Coal Mine Automation project.

## REFERENCES

[1] J.S. Albus, "Outline for a Theory of Intelligence", IEEE Trans. on Systems, Man, and Cybernetics, Vol.21, No.3, May/June 1991

[2] A.J. Barbera., J.S. Albus, M.L. Fitzgerald, and L.S. Haynes, "RCS: The NBS Real-Time Control System," Robots 8 Conference and Exposition, Detroit, MI, June 1984.

[3] Barbera, A.J., J.S. Albus, M.L. Fitzgerald, "Hierarchical Control of Robots Using Microcomputers," Proceedings of the 9th International Symposium on Industrial Robots, Washington, DC, March 1979.

[4] J.S. Albus, "A Theory of Cerebellar Function", Mathematical Biosciences, Vol. 10, pgs. 25-61, 1971.

[5] J.S. Albus, "A New Approach to Manipulator Control : The Cerebellar Model Articulation Controller (CMAC)", *Transactions ASME,* September 1975.

[6] N. Tinbergen, *The Study of Instinct,* Clarendon, Oxford, 1951.

[7] R. Brooks, "A Robust Layered Control System for a Mobile Robot", *IEEE Journal of Robotics and Automation*, vol. RA-2, 1, March, 1986

[8] J.A. Simpson, R.J. Hocken, and J.S. Albus, "The Automated Manufacturing Research Facility of the National Bureau of Standards", *Journal of Manufacturing Systems*, Vol. 1, No. 1, 1983.

[9] J.S. Albus, C. McLean, A.J. Barbera, and M.L. Fitzgerald, "An Architecture for Real-Time Sensory-Interactive Control of Robots in a Manufacturing Environment", *4th IFAC/IFIP Symposium on Information Control Problems in Manufacturing Technology*, Gaithersburg, MD, October 1982.

[10] E. W. Kent and J.S. Albus, "Servoed World Models as Interfaces Between Robot Control Systems and Sensory Data, Robotica, Vol. 2, No.1, January 1984

[11] H.G. McCain, R.D. Kilmer, S. Szabo, A. Abrishamian, "A Hierarchically Controlled Autonomous Robot for Heavy Payload Military Field Applications,"Proceedings of the International Conference on Intelligent Autonomous Systems, Amsterdam, The Netherlands, December 8-11, 1986.

[12] J.S. Albus, "System Description and Design Architecture for Multiple Autonomous Undersea Vehicles", National Institute of Standards and Technology, Technical Report 1251, Gaithersburg, MD, September 1988.

[13] J.S. Albus, H.G. McCain, and R. Lumia, "NASA/NBS Standard Reference Model for Telerobot Control System Architecture (NASREM)," National Institute of Standards and Technology, Technical Report 1235, Gaithersburg, MD , April 1989.

[14] G.N. Saridis, "Foundations of the Theory of Intelligent Controls", *IEEE Workshop on Intelligent Control,* 1985.

[15] G.E. Pugh and G.L. Lucas, "Applications of Value-Driven Decision Theory to the Control and Coordination of Advanced Tactical Air Control Systems", Decision-Science Applications, Inc., Report No. 218, April 1980

[16] L.R. Kelmar, R. Lumia, "World Modeling for Sensory Interactive Trajectory Generation," Proceedings of the Third International Symposium on Robotics and Manufacturing, Vancouver, BC, Canada, July 18-20, 1990.

[17] G. A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information", *The Psychological Review*, 63, pp.71-97, 1956.

[18] A. Meystel, "Theoretical Foundations of Planning and Navigation for Autonomous Robots", *International Journal of Intelligent Systems*, 2, 73-128, 1987

[19] J.E. Laird, A. Newell, and P. Rosenbloom, "SOAR: An Architecture for General Intelligence", *Artificial Intelligence*, vol. 33, 1987, pp. 1-64.

[20] B. Hayes-Roth, "A Blackboard Architecture for Control", *Artificial Intelligence*, pgs. 252-321, 1985.

[21] P. Coad and E. Yourdon, *Object-Oriented Analysis*, Yourdon Press, Englewood Cliffs, New Jersey, 1991

# Appendix B: Strawman Neutral Manufacturing Language for NGC Machine Tool Modules

# Strawman NML
# for NGC Machine Tool Modules

For each level in the NGC control hierarchy, a set of typical task frames have been defined. Following that, a set of machine tool NML message frames are also defined.

Starting with the lowest level in the NGC hierarchy:

## Level 1 - Control Law

### Level 1 (Control Law) machine tool task frames

| | |
|---|---|
| Task Names | SERVO, HI_SERVO, DISCRETE |
| Object | _____ |
| Goal | _____ |
| Parameters | _____ |
| Agents | Axis actuators, or discrete actuators |
| Requirements | Axis limits ok, discrete combinations ok |
| Procedures | Control algorithm (e.g. PD, PID, etc.), discrete delays, etc. |
| | Trajectory point interpolation algorithm (e.g. straight line, polynomial, etc.) |

### Level 1 NML message frames

**Control Law COMMAND Input from Level 2 Profile Generator**

| | |
|---|---|
| Command Name | SERVO |
| Object | Cutting tool id |
| Goal | Desired pose (i.e. position and orientation), velocity, acceleration, jerk, force, or force-rate of tool tip in part coordiinates |
| Parameters | Command ID (index that uniquely defines a specific command) |

**Control Law COMMAND Input from Level 1 Operator**

| | |
|---|---|
| Command Name | HI_SERVO |
| Object | (Same as from Level 2 profile generator) |

| | |
|---|---|
| Goal | (Same as from Level 2 profile generator) |
| Parameters | (Same as from Level 2 profile generator)<br>Operating mode (e.g. Manual, shared, automatic) |

**REPLY (STATUS) Output to Level 2 Profile Generator**

| | |
|---|---|
| Command ID | ID of currently executing command |
| Status | Status of current command execution |
| Operating mode | Manual, shared, or automatic |

**REPLY (STATUS) Output to Level 1 Operator**

Same as REPLY output to level 2 profile generator

**QUERY Output to Level 1 World Model**

Request for RETRIEVAL input from level 1 world model. (This request may be implicit.)

**RETRIEVAL Input from Level 1 World Model**

State clock and sync signals
Observed state vector defining the observed pose (i.e. position and orientation), velocity, acceleration, jerk, force, or force-rate of each actuator.
Jacobian matrix for inverse kinematic coordinate transform (tool => actuator)

**COMMAND Output to Sensor / Actuator Bus**

Desired actuator force, torque, velocity, or power

## Level 2 - Profile Generator

### Level 2 (Profile Generator) typical task frames

| | |
|---|---|
| Task Names | DYNAMIC_PATH, or OPERATOR_DYNAMIC-PATH |
| Object | _____ |
| Goal | _____ |
| Parameters | _____ |
| Agents | Servo controllers |
| Requirements | Load dynamics within capacity of actuators |
| Procedures | Acceleration and deceleration algorithms for computing dynamic trajectories |

### Level 2 NML frames

**Profile Generator COMMAND Input from level 3 Machine Executive**

| | |
|---|---|
| Command Name | DYNAMIC_PATH |
| Object | Cutting tool |
| Goal | Desired pose (i.e. position and orientation), velocity, acceleration, jerk, force, or force-rate of tool |
| Parameters | Command ID<br>Goal pose tolerance<br>Dynamic path deviation tolerance<br>Trajectory dynamics objective function<br>Command coordinate system (= part feature)<br>Position/Force selection vector<br>Redundancy resolution specification |

**COMMAND Input from level 2 operator**

| | |
|---|---|
| Command Name | OPERATOR_DYNAMIC_PATH |
| Object | (Same as from Level 3 machine executive) |
| Goal | (Same as from Level 3 machine executive) |
| Parameters | (Same as from Level 3 machine executive)<br>Operating mode (e.g. Manual, shared, automatic) |

**REPLY Output to Level 3 Machine Executive**

| | |
|---|---|
| Command ID | ID of currently executing command |
| Status | Status of current command execution<br>Estimated execution termination time |
| Operating mode | Manual, shared, or automatic |

**REPLY Output to Level 2 Operator**

Same as reply output to level 3 machine executive

**QUERY Output to Level 2 World Model**

Requests for information RETRIEVAL from level 2 World Model

**RETRIEVAL Input from Level 2 World Model**

State clock and sync signals
Observed pose, velocity, force of load or tool
Trajectory segment defining observed poses, velocities, and forces of load or tool
Jacobian matrix for inverse kinematic and force coordinate transform (part feature => tool)
Manipulator dynamic model
Load dynamic model
Object contact point parameters (position, velocity, stiffness, friction, strength, etc.)
Limits of travel in coordinate system of command
Singularity clearance margins
Position, velocity, and force limits
Estimated time or distance to contact

**COMMAND Output to Level 1 Control Law**

Same as level 1 COMMAND input from level 2 profile generator

## Level 3 - Machine Executive

### Machine Executive level task frames

| | |
|---|---|
| Task Names | MOVE-TO, MACHINE-FEATURE, etc. |
| Object | _____ |
| Goal | _____ |
| Parameters | _____ |
| Agents | Dynamically coupled motion controllers |
| Requirements | Enabling conditions (conditions that must be satisfied to begin or continue the commanded task) |
| | Disabling conditions (conditions that will cause the commanded task to be interrupted or discontinued |
| Procedures | Cutter location path planning algorithms for machining features, including checks to detect and avoid collisions with fixtures or other obstacles |

### Machine Executive level NML frames

**COMMAND Input from Level 4 Task Coordinator**

| | |
|---|---|
| Command Name | MOVE-TO, MILL-FEATURE, DRILL-FEATURE, GRIND-FEATURE, etc. |
| Object | ID of part surface or feature being machined |
| Goal | Desired path of tool relative to part in part coordinates |
| Parameters | Command ID |
| | Goal condition tolerance |
| | Tool to be used |
| | Axis coordination requirements |
| | Speed, or completion time requirements |

**COMMAND Input from level 3 operator**

(Same as COMMAND Input from Level 4 task coordinator), plus Operating mode (e.g.

Manual, shared, automatic)

**REPLY output to level 4 task coordinator**

Operating mode

Planning queue status

Execution command ID

Execution status

Estimated execution termination time

**REPLY output to level 3 operator**

Same as reply output to level 4 task coordinator

**QUERY output to level 3 world model**

Request for information RETRIEVAL input from level 3 world model

**RETRIEVAL input from level 3 world model**

State clock and sync signals

Jacobian matrix for inverse coordinate transform (object => object surface or feature)

Machine tool geometry and dynamic model

Object machinability parameters

Limits of travel in coordinate system of command

Position, velocity, and force limits

Estimated obstacle clearance for planned paths

Cost/benefit analysis on hypothesized path plans

**COMMAND output to level 2 profile generator**

Same as level 2 command input from level 3 machine executive

# Level 4 - Task Coordinator

## Level 4 (Task Coordinator) task frames

| | | |
|---|---|---|
| Task Names | | MACHINE-PART, GRIND-PART, DEBURR-PART, etc. |
| | Object | _____ |
| | Goal | _____ |
| | Parameters | _____ |
| | Agents | Machine subsystems such as motion control, tool changers, coolant spray, etc. |
| | Requirements | Tools or fixtures to be used |
| | | Task enabling and disabling conditions |
| | | Task synchronization and scheduling requirements |
| | Procedures | Process plans or planning algorithms for generating sequences machine executive commands |

## Level 4 Task Coordinator NML frames

**COMMAND Input from Level 5 Workstation Executive**

| | | |
|---|---|---|
| Command Name | | MACHINE-PART, GRIND-PART, DEBURR-PART, etc. |
| Object | | ID of part or material stock to be acted upon (as-is condition) |
| Goal | Either | 1) Desired task or operation to be performed on an object. |
| | Or | 2) Goal state to be achieved by operation (to-be condition) |

(Note: Many Task Coordinator commands at level 4 may have names similar or identical to Machine Executive commands at level 3. The difference is that, at level 4, the object acted upon is a part with many surfaces or features, whereas at level 3, the object is a single surface or feature of the part.)

| | |
|---|---|
| Parameters | Command ID |
| | Goal state tolerances |
| | Coordinate system (=workspace machine tool) |
| | Task or part priority |
| | Coordination requirements with other machines |

Duration, speed, or completion time requirements

(Note: At level 4 and above, an input list containing more than one task command at a time may be permitted. This list should be prioritized, and must specify sequential ordering requirements if they exist.)

**REPLY output to level 5 Workstation Executive**

Operating mode

Planning command ID

Planning queue status

Execution command ID

Execution status

Estimated execution termination time

**COMMAND input from level 4 operator**

(Same command input as from level 5 Workstation Executive), plus

Operating mode Manual, shared, or automatic

**REPLY output to level 4 operator**

(Same as report output to level 5 Workstation Executive)

**QUERY output to level 4 world model:**

(Request for RETRIEVAL input from level 4 world model)

**RETRIEVAL input from level 4 world model**

State clock

Sync signals from other equipment

Observed degree of task completion

Workspace geometry and dynamic model

Workspace attributes (lighting, temperature, visibility, etc.)

Resources available to machine tool (parts, tools, fixtures, etc.)

Measured part geometry

Measured part state variables (position, orientation, velocity)

Part and tool attributes (color, mass, hardness, surface characteristics)

State of enabling and disabling conditions

Predicted results of hypothesized plans

Cost/benefit evaluation of hypothesized plan results

**COMMAND output to level 3 Machine Executive**

(Same as level 3 command input from level 4 Task Coordinator)

## Level 5 - Workstation

### Level 5 Workstation task frames

| | |
|---|---|
| Task Names | MACHINE-GROUP (of parts in a tray, for example), etc. |
| Object | _____ |
| Goal | _____ |
| Parameters | _____ |
| Agents | Machine tools, robots, part buffers, inspection devices, etc. |
| Requirements | List of tools, materials, and fixtures to be used |
| | Task enabling and disabling conditions |
| | Task synchronization requirements with other equipment |
| Procedures | Process plans or planning algorithms for generating coordinated sequences of level 4 task coordinator commands |

### Level 5 Workstation NML frames

**COMMAND Input from Level 6 Cell Control**

| | |
|---|---|
| Command Name | MACHINE-GROUP, etc. |
| Object | ID of group of objects to be acted upon (as-is) |

| | | |
|---|---|---|
| Goal | Either | 1) List of workstation tasks to be performed on a group of objects |
| | Or | 2) List of goal states to be achieved (to-be) |
| Parameters | | Command ID |
| | | Goal condition tolerances |
| | | Coordinate system (= workstation) |
| | | Group priority |
| | | Completion time requirements |

**REPLY output to level 6 Cell control**

Operating mode

Planning command ID

Planning queue status

Execution command ID

Execution status

Estimated execution termination time

**COMMAND input from level 5 operator**

(Same inputs as command inputs from level 6 Cell control), plus

Operating mode (shared, manual, or automatic)

**REPLY output to level 5 operator**

(Same as reply output to level 6 Cell control)

**QUERY output to level 5 world model**

Request for RETRIEVAL feedback from level 5 world model

**RETRIEVAL input from level 5 world model**

State clock

Sync signals from other equipment

Observed degree of task completion

Workspace geometry and task timing model

Equipment availability in workstation

Workstation equipment capabilities

Tray state variables (tray pose, tray slot occupancy)

Tray attributes (size, shape, bar code)

State of enabling and disabling conditions

Predicted results of hypothesized plans

Cost/benefit evaluation of hypothesized plan results

**COMMAND output to level 4 Task Coordinator**

(Same as level 4 command input to from level 5 Workstation Executive)

# Appendix C: Strawman NGC Commands for 3-axis Machining

# NGC COMMANDS FOR 3-AXIS MACHINING

T. Kramer

November 4, 1991

This is a list of proposed NGC commands for a 3-axis machining center. Most of these commands should be suitable for machining centers with more than three axes. Commands at level 4 (task) level 3 (E-move), and level 2 (primitive) of the NASREM hierarchy are given.

For each command, the name of the command and a description of the effects of the command are given. Parameter names are given in italics for all commands. Allowable values for parameters are not specified here but need to be provided. In most cases the nature of the parameter value is evident from the parameter name. It would be very desirable to use a formal information modeling language such as EXPRESS to specify these commands in more detail. Among other desirable features, EXPRESS provides a rich method of describing allowable parameter values.

The capability to handle expressions will be desirable in NGC controllers. To handle expressions, there must be a set of built-in operations (sum, square root, logical and, sine, etc.). Expressions are used by permitting a command to contain an expression in lieu of any parameter value, so long as each expression yields the proper type of value when it is evaluated. Immediately prior to executing a command, the controller evaluates any expressions in the command. Some existing controllers (GE-2000, for example) have the capability to handle expressions. It is assumed here that expressions will be allowed in NGC controllers, but no definitions of expressions have been adopted. A specification of expressions will be required. It seems desirable to have the expression rules be part of a command language which may be applied to all levels of the NGC controller, rather than to adopt a different set of rules for expressions at each level.

## Level 4 Task

REMOVE_VOLUMES *plan_id, design_id, material_removal_volumes_id, setup_id, workpiece_id, fixture_id*

The REMOVE_VOLUMES command causes a set of material removal volumes to be removed from a workpiece. The command parameters include a process plan identifier (*plan_id*), a design identifier (*design_id*), an identifier for the set of material removal volumes referenced in the program (*material_removal_volumes_id*), an identifier for setup instructions (*setup_id*), an identifier for the workpiece that will be machined (*workpiece_id*), and an identifier for the fixture to be used (*fixture_id*). This command is used to do the work done in a single fixturing of a workpiece. The workpiece may start as a piece of stock or as a partially machined workpiece. The workpiece may end as a completely machined part or as a partially machined workpiece.

## Level 3 E-move

For those level 3 commands which require a material removal volume, it is expected that the volume will be of a type described in a library of generic removal volumes as being suitable for use with the operation. When an operation is finished, no material may remain in the material removal volume, and the operation may remove no material outside the material removal volume. Material removal volumes are not necessarily completely filled with material before the operation starts. A proposed library of material removal volumes has been defined at NIST. Material removal volume definitions referenced in level 3 commands would be in the world model.

The level 3 (E-move) commands may be arranged in a hierarchy as shown on the last page of this paper. The descriptions of the individual commands do not refer to the hierarchy.

---

BORE *tool_type_id, material_removal_volume, spindle_speed, feed_rate*

The BORE command results in a hole being bored. The cutter must be a boring tool.

---

CENTER_DRILL *tool_type_id, material_removal_volume, spindle_speed, feed_rate*

The CENTER_DRILL command results in a small starter hole being made with a center_drill cutter by a single-stroke plunge into the material.

---

COUNTERBORE *tool_type_id, material_removal_volume, spindle_speed, feed_rate*

The COUNTERBORE command results in an existing hole being enlarged.

---

END_PROGRAM *(no parameters)*

The END_PROGRAM command indicates the end of a program has been reached. It may cause activities such as spindle retract, return to home position, cleanup of the world model, resetting machine parameters, etc.

---

FACE_MILL *tool_type_id, material_removal_volume, spindle_speed, feed_rate, pass_depth,*

*stepover*

The FACE_MILL command results in the material removal volume being machined away by a face_mill cutter.

---

FINISH_MILL *tool_type_id, material_removal_volume, spindle_speed, feed_rate, stepover*

The FINISH_MILL command results in the removal with a finish end_mill (with cutter nose geometry suitable for the material removal volume) of any material in the material removal volume, so that the resulting surfaces meet some desired quality specification. Only a small thickness of material should be removed in this operation. The *stepover* parameter is required for milling with the flat portion of the nose of the cutter, where an area larger than the area of the nose of the tool is being finished.

---

FLY_CUT *tool_type_id, material_removal_volume, spindle_speed, feed_rate, pass_depth, stepover*

The FLY_CUT command results in the material removal volume being machined away by a fly_cutter.

---

INITIALIZE_PROGRAM *program_name, design_id, material_removal_volumes_id, setup_id, workpiece_id, material, fixture_id, program_x_zero, program_y_zero, program_z_zero*

The INITIALIZE_PROGRAM command initializes the controller to be ready to accept additional level 3 commands, all of which, up to an END_PROGRAM command, are logically parts of a single program for machining a single workpiece using a single fixture. The command identifies the name of the program (*program_name*), a design identifier (*design_id*), an identifier for the set of material removal volumes referenced in the program (*material_removal_volumes_id*), an identifier for setup instructions (*setup_id*), an identifier for the workpiece that will be machined (*workpiece_id*), the name of the type of material being machined (*material*), an identifier for the fixture to be used (*fixture_id*), and the location of the program zero in machine coordinates (*program_x_zero, program_y_zero,* and *program_z_zero*). This command causes no motion in the machining center. This command is not used to bring the machining center task controller to a ready state from a cold start; that must be done before an INITIALIZE_PROGRAM command is given.

---

MACHINE_CHAMFER *tool_type_id, material_removal_volume, spindle_speed, feed_rate*

The MACHINE_CHAMFER command results in an edge being chamfered. The cutter must be a chamfer tool (tool profile is a cone, possibly truncated).

---

MACHINE_COUNTERSINK *tool_type_id, material_removal_volume, spindle_speed, feed_rate*

The MACHINE_COUNTERSINK command results in a hole being countersunk with a countersink cutter.

---

MACHINE_ROUND *tool_type_id, material_removal_volume, spindle_speed, feed_rate*

The MACHINE_ROUND command results in an edge being rounded. The cutter must be a rounder (side of tool profile is an arc of a circle).

---

PERIPHERAL_MILL *tool_type_id, material_removal_volume, spindle_speed, feed_rate, pass_depth, stepover*

The PERIPHERAL_MILL command is for milling an exterior or interior contour by milling at the periphery only. Unlike rough_mill, it may not contain any plunging or slotting. The cutter must be an end_mill (with nose geometry suitable for the material removal volume).

---

REAM *tool_type_id, material_removal_volume, spindle_speed, feed_rate*

The REAM command causes a small amount of material to be removed from the inside of an existing hole. The cutter must be a ream. The material cut away must be a very small thickness around the surface of the material removal volume.

---

ROUGH_MILL *tool_type_id, material_removal_volume, spindle_speed, feed_rate, pass_depth, stepover*

The ROUGH_MILL command results in the milling with an end_mill of the designated material removal volume. It is expected that requirements on the surfaces created by this operation will be such that no consideration needs to be given to surface quality in determining machining methods. The operation may include plunging, slotting, and both conventional cutting and climb cutting. The cutter which is used may be a rough end_mill or a finish end_mill (with nose geometry suitable for the material removal volume).

---

SET0_CENTER *tool_type_id, near_x, near_y, x_offset, y_offset, near_diameter*

The SET0_CENTER command results in a probe cycle being run in which a hole with its axis parallel to the z-axis is probed. Program x_zero and y_zero are set at the center of the hole or by offsetting from the center. The tool must be a probe.

---

SET0_CORNER *tool_type_id, near_x, near_y, x_offset, y_offset, corner_type*

The SET0_CORNER command results in a probe cycle being run in which a corner is probed. The corner must be formed by two planes parallel to the z-axis. Program x_zero and y_zero are set at the corner or by offsetting from the corner. The tool must be a probe.

---

SET0_Z *tool_type_id, x_location, y_location, z_offset*

The SET0_Z command results in a probe cycle being run in which a surface parallel to the xy-plane is probed. Program z_zero is set at the surface or by offsetting from the surface. The tool must be a probe.

---

SLOT_MILL *tool_type_id, material_removal_volume, spindle_speed, feed_rate, pass_depth*

The SLOT_MILL command results in a slot being milled. The shape of the material removal volume will be such that it may be produced by having the tool follow a path (simple or complex) in which the tool will generally be cutting across its full width to form a slot.

---

TAP *tool_type_id, material_removal_volume, spindle_speed, feed_rate*

The TAP command results in the inside of an existing hole being threaded. The cutter must be a tap.

---

TWIST_DRILL *tool_type_id, material_removal_volume, spindle_speed, feed_rate, pass_depth*

The TWIST_DRILL command results in a hole being drilled. The *pass_depth* parameter is used if the user's TWIST_DRILL strategy is to perform peck drilling. It may be desirable to split this command into three commands: plunge_drill, peck_drill, and small_hole_drill.

---

## Level 2 Primitive

This set of primitive commands is intended to provide for all the generic primitive functions of popular existing machine tool controllers for single-spindle 3-axis machining centers (vertical or horizontal). The primitive capabilities of a Monarch VMC-75 vertical machining center with a GE-2000 controller and a Renishaw probe are covered here, with the exceptions noted below, though the commands here have different names and often somewhat different effects. It will be very desirable to check this set against additional existing machine tool plus controller combinations.

Two desirable capabilities (2D and 3D parametric tool motion) that the GE-2000 does not have but could be performed on the Monarch VMC-75 with a different controller are included here.

GE-2000 capabilities to execute macro commands (instructions G61, G79, and G81 through G89) are not covered here because they are conceptually E-move level commands. GE-2000 capabilities for setting registers are not covered here because it is expected they will be covered by provisions for handling expressions discussed at the beginning of this paper. GE-2000 "W-axis" commands for moving the head of the machining center are not covered here because the conceptual view taken here of a 3-axis machining center does not require a movable head. The GE-2000 commands for switching between incremental and positional interpretation of location data (g90 and g91) are not covered because this represents changing the command interpreter. The following GE-2000 commands are not covered for a variety of reasons: (G21, G22, G23 - conceptually redundant with straight line and arc motion), (G51), (G57), (G77, G78, G80, G94 - not needed), (G92), (G93 - redundant with setting feed rate), (many M commands - special-purpose machine-specific capabilities).

The commands required for a using a touch probe in the spindle of the machining center are included in this set. The probe is being treated as part of the machining center. An alternative method of dealing with the touch probe would be to treat it as a separate device with its own controller. Other sensory devices are not dealt with here. If it is decided to treat sensory devices as part of the machining center, sets of commands for such devices will have to be added to this set. If sensory devices are to be handled with separate controllers, the touch probe commands here might be deleted.

For several pairs (e.g., FLOOD_ON and FLOOD_OFF) or sets of commands (e.g., USE_NO_TOOL_LENGTH_OFFSETS, USE_MODIFIED_TOOL_LENGTH_OFFSETS, USE_NORMAL_TOOL_LENGTH_OFFSETS), it would be feasible to replace the pair or set with a single command having a parameter. For example there could be a SET_FLOOD command whose parameter could take the values "on" and "off", and there could be a SET_TOOL_OFFSET_MODE command whose parameter could have the values "no_offset", "normal" or a register number (for the "modified" mode). In general, separate commands have been kept here, since separate commands seem easier to use and less likely to be used incorrectly, but consideration might be given to replacing them.

**Activities that Control Spindle Translation**

It is assumed in these activities that the spindle tip is always at some location called the "current location", and the controller always knows where that is. It is also assumed that there is always a "selected plane" which must be the xy-plane, the yz-plane, or the xz-plane of the machine.

---

ARC_FEED *first_axis_coordinate, second_axis_coordinate, rotation, axis_end_point*

Move in a helical arc from the current location at the existing feed rate. The axis of the helix is parallel to the x, y, or z axis, according to which one is perpendicular to the selected plane. The helical arc may degenerate to a circular arc if there is no motion parallel to the axis of the helix. If the selected plane is the xy-plane, *first_axis_coordinate* is the axis x-coordinate, *second_axis_coordinate* is the axis y coordinate, and *axis_end_point* is the z-coordinate of the end of the arc. If the selected plane is the yz-plane, *first_axis_coordinate* is the axis y-coordinate, *second_axis_coordinate* is the axis z-coordinate, and *axis_end_point* is the x-coordinate of the end of the arc. If the selected plane is the xz-plane, *first_axis_coordinate* is the axis x-coordinate, *second_axis_coordinate* is the axis z coordinate, and *axis_end_point* is the y-coordinate of the end of the arc. The *rotation* parameter represents the number of degrees or radians in the arc. *Rotation* is positive if the arc is traversed counterclockwise as viewed from the positive end of the coordinate axis perpendicular to the currently selected plane. The radius of the helix is determined by the distance from the current location to the axis of helix.

There are several ways in which the parameters for ARC_FEED could have been selected. Also, the command could have been split into two commands, one for clockwise and the other for counterclockwise, as is currently done with G2 and G3 codes in the RS-274D standard. The parameters selected above have the advantages that (1) there is no redundant information in the parameter values, (2) any set of numerical parameter values makes sense, and (3) any arc of a given helix (or several passes around a circle) can be produced by a single command (a capability existing controllers generally do not provide).

One alternative used by existing controllers is to specify the radius of the helix and the end point of the arc. In this parameterization it is assumed that the arc is not more than 360 degrees. This set of parameters allows two possible arcs, a large one and a small one, so a method of discriminating between the two is required, either by an additional parameter, or by overlaying an additional discriminator on the existing parameters, such as having the radius be negative for a large arc and positive for a small one (this is done in some existing systems). In this parameterization it is possible to give parameter values which do not make sense, since only those points within a cylinder whose axis passes through the current location and whose radius is twice the given radius value can be reached by the arc.

---

DWELL *duration*

Do not move for the amount of time specified by *duration.* The ability to dwell is useful in finish cutting.

---

PARAMETRIC_2D_CURVE_FEED *first_function, second_function, start_parameter_value, end_parameter_value*

Move along a parametric curve in the selected plane. We will call the parameter *u.* If the selected plane is the xy-plane, *first_function* gives x in terms of *u* and *second_function* gives y in terms of *u.* Analogous assignments are made if the selected plane is the xz-plane or the yz-plane. Allowable functions should include at least cubic polynomials and elementary trigonometric functions. The current position of the spindle must be at coordinates corresponding to the *start_parameter_value.* The final position of the spindle is at coordinates corresponding to the *end_parameter_value.*

---

PARAMETRIC_3D_CURVE_FEED *x_function, y_function, z_function, start_parameter_value, end_parameter_value*

Move along a parametric curve in three dimensions, where x, y, and z are each functions of the parameter *u.* Allowable functions should include at least cubic polynomials and elementary trigonometric functions. The current position of the spindle must be at the coordinates corresponding to the *start_parameter_value.* The final position of the spindle is at the coordinates corresponding to the *end_parameter_value.*

---

SPINDLE_RETRACT *(no parameters)*

Retract at traverse rate to fully retracted position.

---

STRAIGHT_FEED *x, y, z, probe*

Move in a straight line at existing feed rate (or using the existing z-force) from the current point to the point given by the *x, y* and *z* parameters. If z-force is enabled, the values of *x* and *y* must be the same as those of the current point.

The *probe* parameter is optional. If the *probe* parameter is present, the feed motion will stop when the probe is tripped or when the endpoint is reached, whichever happens first. The probe must be in the spindle and turned on beforehand if the *probe* parameter is used. If the *probe* parameter is present and the probe is tripped during the motion, the fact that the probe was tripped and the

position of the spindle at the time the probe was tripped will be recorded in the appropriate registers or variables.

If the probe is tripped, the end position of the spindle is not determined, except that it must lie on the straight line being followed, must lie between the trip point and the programmed end point, and must be within a certain distance of the trip point. This indeterminacy is necessary because the spindle cannot be stopped instantly and the controller cannot look ahead and slow it down as it nears the end, as it would normally do for other motions. It should be feasible to require the stopping point to be a fixed distance from the trip point, if that seems more useful.

It may be useful to define a separate command for using the probe. It may be useful to allow probing while following other paths.

---

STRAIGHT_TRAVERSE *x*, *y*, *z*

Move in a straight line at traverse rate from the current point to the point given by the *x*, *y* and *z* parameters.

---

**Other Physical Activities**

Mist coolant and flood coolant are covered here. Through-spindle coolant might be added.

---

CHANGE_TOOL *slot_number*

It is assumed that each cutting tool in the machine is assigned to a slot (intended to correspond to a slot number in a tool carousel). This command results in the tool currently in the spindle (if any) being returned to its slot, and the tool from the slot designated by *slot_number* being inserted in the spindle. If there is no tool in the slot designated by *slot_number*, there will be no tool in the spindle after this command is executed. For the purposes of this command, the tool includes the tool holder.

It may be desirable to add a command something like "change_tool_buffer" for machines which have a tool change buffer. Because of the time saved by a tool change buffer, it seems likely that more machining centers will have them in the future.

---

FLOOD_OFF *(no parameters)*

Turn flood coolant off.

---

FLOOD_ON *(no parameters)*

Turn flood coolant on.

---

LOCK_SPINDLE_Z *(no parameters)*

Lock the spindle against vertical motion. This has no logical effect required for control, so it might be deleted. Whether there is a spindle lock is an implementation detail.

---

MIST_OFF *(no parameters)*

Turn mist coolant on.

---

MIST_ON *(no parameters)*

Turn mist coolant on.

---

ORIENT_SPINDLE *orientation direction*

This commands causes the spindle to turn at the current spindle speed in the given *direction* (clockwise or counterclockwise) to the given *orientation* and stop. This command can be given only if the spindle is not turning beforehand.

---

START_SPINDLE_CLOCKWISE *(no parameters)*

Turn spindle clockwise at currently set speed rate.

---

START_SPINDLE_COUNTERCLOCKWISE *(no parameters)*

Turn spindle counterclockwise at currently set speed rate.

---

STOP_SPINDLE_TURNING *orientation*

Stop spindle from turning. The *orientation* parameter is optional and indicates the orientation of the spindle when it stops. If the spindle is already stopped, this command may be given as long as there is no value for *orientation.*

---

TURN_PROBE_OFF *(no parameters)*

Turn the probe off.

---

TURN_PROBE_ON *(no parameters)*

Turn the probe on.

---

UNLOCK_SPINDLE_Z *(no parameters)*

Unlock the spindle to permit vertical motion. This has no logical effect required for control, so it might be deleted. Whether there is a spindle lock is an implementation detail.

---

**Data Activities**

SET_CUTTER_RADIUS_COMPENSATION *radius*

Set the radius value to be used in cutter radius compensation.

---

SET_FEED_RATE *feed_rate*

Set the feed rate that will be used when the spindle is told to move at the currently set feed rate.

---

SET_SPINDLE_FORCE *force*

Set the force with which the spindle is pushed in the z-direction. The force is positive when it is in the positive z-direction. The force is negative when it is in the negative z-direction. This setting is used for motions parallel to the z-axis whenever the USE_SPINDLE_FORCE command is in effect.

---

SET_SPINDLE_SPEED *speed*

Set the spindle speed that will be used when the spindle is turning. This is usually given in rpm and refers to the rate of spindle rotation.

---

SET_TRAVERSE_RATE *rate*

Set the traverse rate that will be used when the spindle traverses. It is expected that no cutting will occur while a traverse move is being made.

---

**Control Activities**

CANCEL_SPEED_FEED_SYNCHRONY *(no parameters)*

Do not require spindle speed and feed rate to be exactly synchronous.

---

SELECT_PLANE *selected_plane*

Use the plane designated by *selected_plane* as the selected plane.

---

START_CUTTER_RADIUS_COMPENSATION *side*

Apply cutter radius compensation when executing spindle translation commands. The value stored in the cutter radius register will be used. The *side* parameter indicates whether the cutter is to be to the left or right of the path described by a command.

---

STOP_CUTTER_RADIUS_COMPENSATION *(no parameters)*

Do not apply cutter radius compensation when executing spindle translation commands.

---

SYNCHRONIZE_SPEED_AND_FEED *(no parameters)*

The spindle speed and feed rate on a machining center are not normally forced to bear an exact relation to one another, because it is usually more useful to be able to control them independently (so that they may be accelerated and decelerated independently, for example). However, in tapping, exact synchrony is required. This command tells the controller to maintain exact synchrony, insofar as possible.

---

USE_ABSOLUTE_ORIGIN *(no parameters)*

Use the absolute machine origin.

---

USE_LENGTH_UNITS *units*

Use the specified *units* for length. The possible value of *units* should include inches, centimeters, and possibly others (millimeters and meters are strong candidates). It might be useful to have a similar command for angle units, to allow switching between degrees and radians.

---

USE_MODIFIED_TOOL_LENGTH_OFFSETS *register*

Use the tool length offsets given in the tool length offset registers as modified by the number in the given *register*. This command covers modifying tool length offsets to compensate for different fixtures or for moving the head of the machining center (if it has a movable head). It may be desirable to allow other parameter values than a register number.

---

USE_NORMAL_TOOL_LENGTH_OFFSETS *(no parameters)*

Use the tool length offsets given in the tool length offset registers.

---

USE_NO_SPINDLE_FORCE *(no parameters)*

Do not use spindle force. Instead, use the feed rate to determine spindle motion.

---

USE_NO_TOOL_LENGTH_OFFSETS *(no parameters)*

Do not use tool length offsets.

---

USE_PROGRAM_ORIGIN *(no parameters)*

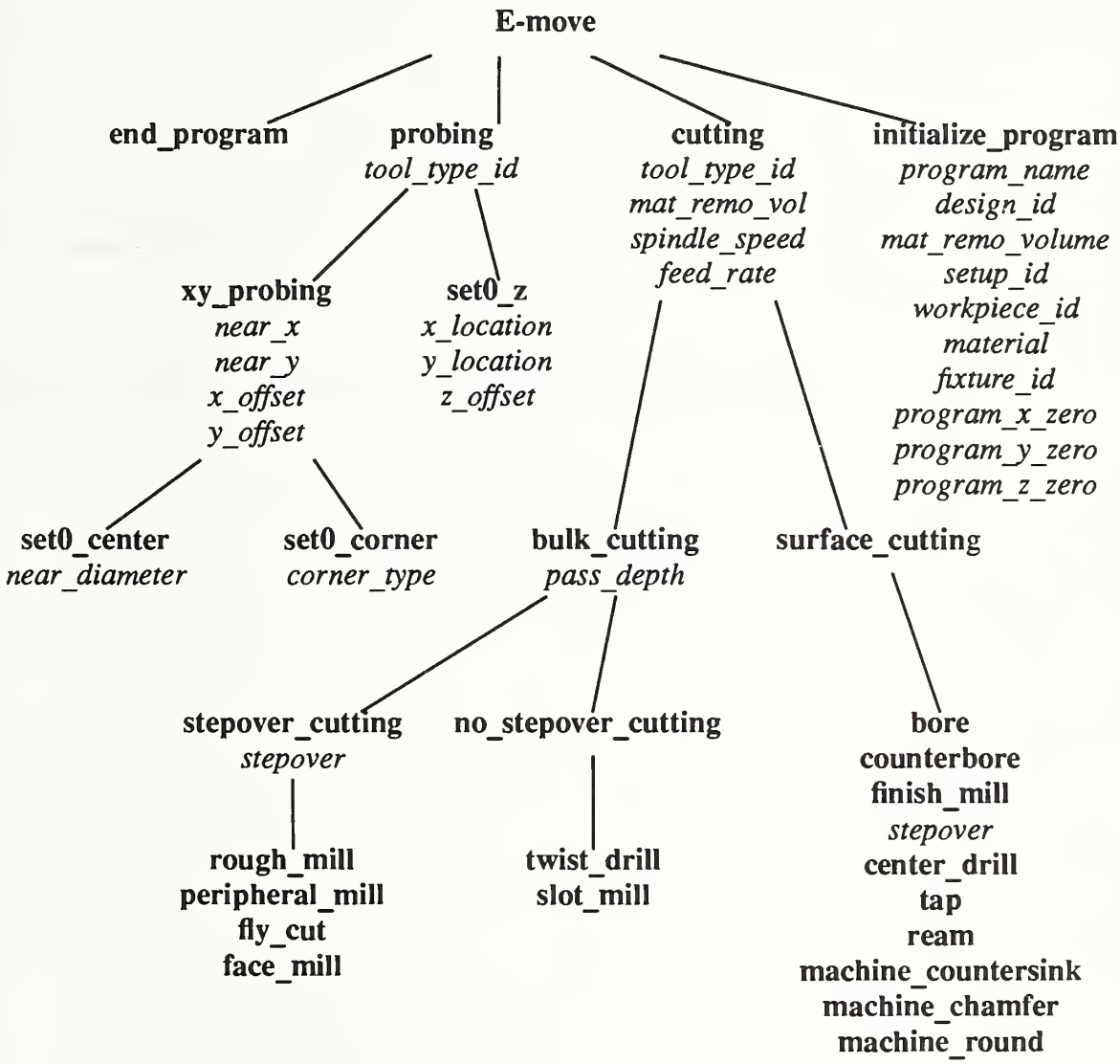
Use the origin specified in the program origin register.

---

USE_SPINDLE_FORCE *(no parameters)*

This is useful in tapping, where a tap may be pushed into a hole with moderate force and screw itself in. Then a moderate force is exerted out of the hole and the tap screws itself out again. Tapping this way makes it unnecessary to coordinate axial motion with tap rotation, which is required by the other standard tapping method.

---




## Hierarchy of Machining Operations - E-move Level

machining operations are shown in **boldface** type
attributes are shown in *italic* type
Only leaf nodes of the hierarchy may be instantiated.
To find all the attributes of an operation, trace down the hierarchy from "E-move" to the operation and include the attributes of every node along the path.

# Appendix D: ADACS System Task Analysis

# ADACS SYSTEM TASK ANALYSIS

Introduction

This appendix presents a task frame analysis of the Advanced Deburring and Chamfering System (ADACS). The ADACS is an experimental deburring and chamfering system under development at the National Institute of Standards and Technology. The ADACS highlights technological progress in advanced sensory-interactive, robot controlled tooling. The ADACS workstation is composed of a Cincinnati Milicron T3 robot, a Kinetics ADT chamfering tool, a programmable logic controller for an array of tools, and a fixturing vise. At present, the ADACS requires an operator to deliver parts, fixture individual parts in the vise, initiate the automated chamfering and remove the chamfered part.

The emphasis of this document is on the system representation of the ADACS. While analyzing the ADACS, the concept of *task frame* is used to organize the representation of ADACS system - which includes data, functionality and application to a set of predefined tasks. The analysis uses the NASREM hierarchical levels of control to stratify the tasks across a temporal and spatial domain. Within the NASREM hierarchy, an ADACS level-by-level task vocabulary is presented that encapsulates the actions of the goal-driven system. Given the set of ADACS task verbs, each verb as applied to a class of objects is then studied. For the ADACS project the objects consisted of hard metal parts - either titanium or inconel. Given instances of the verb-object pairings, task frames are filled with knowledge derived from the analysis of the data and procedural knowledge that is necessary to achieve the verb goals. Specific categories of task behavior and information modelling are enumerated within the task frame methodology. The rest of this document details the results of the task analysis of the ADACS system in the task frame format.

Comments:

The ADACS task frames presented herein reasonably cover most of the ADACS functionality. At this point, the notation and terminology are at times inconsistent. A more formal task frame description language - such as EXPRESS - is under review.

Appendix Contents:

1) ADACS Command Vocabulary for each Level of the NASREM Hierarchy

2) Generic Task Frame Template

3) ADACS Set of Task Frames

4) Abridged Lexicon of ADACS terminology

# ADACS COMMAND VOCABULARY for each Level of the NASREM Hierarchy

## Level 5 Workstation - INPUT

INIT plans the sequence of tasks required to put the ADACS workcell into ready status.

CHAMFER-BATCH plans the sequence of acquisition/delivery tasks and chamfering tasks required to chamfer a batch of parts.

SHUTDOWN plans the sequence of tasks required to put the ADACS into safe and idle status.

## Level 4 Task

INIT plans the sequence of tasks required to put the agents of the Task Level and lower levels into ready status.

CHAMFER-PART takes a PDES description of a part, knowledge of the finishing requirements, and the location of the part, and plans a sequence of part fixturing and edge chamfering operations for a single part.

SHUTDOWN-TASK plans the sequence of tasks required to put the agents of the Task Level and lower levels into safe and idle status.

FIXTURE-PART* plans the sequence of operations required to fixture the part in the vise. This command requires knowledge of the beginning location of the part in the workcell and the final location of the part in the vise.

UNFIXTURE-PART * plans the sequence of operations required to remove the part from the vise. This command requires that the final location of the part be specified.

## Level 3 Elemental Move

INIT plans the sequence of operations required to put the E-Move Level and lower levels into ready status.

EMOVE->T3.MOVE plans the collision-free path from the current pose of the T3 robot to the given goal pose.

CHAMFER-FEATURE plans the sequence of robot and tool commands required to approach the feature, activate the tool, track the feature, and depart from the feature. The geometry of the feature is defined in PDES format, the finishing requirements for the feature are also defined, and the position of the part in the vise is known.

SHUTDOWN* plans the sequence of operations required to put the E-Move Level and lower levels into safe and idle status.

## Level 2 Primitive - T3

T3.INIT plans the sequence of steps required to power up the T3 robot and establish the communication link between it and its host computer.

T3.MOVE takes a list of intermediate poses, and plans a trajectory through those poses. Each intermediate move can be either straight, circular, or a B-spline. Motion between each intermediate pose is smoothed such that the velocity of the tool center point is non-zero between any successive poses, and zero at the end. Parameters for each successive move differ depending upon the nature of the move. Straight moves require that the speed and acceleration be specified or are set to some default. Circular moves require a parameter to specify the circle (such as center point or a third point on the circle). Spline moves require spline information, such as the knot points, time for motion, etc.

T3.MOVE-UNTIL takes a single pose, and will move in either a straight, circular, or B-spline path until a prescribed force is attained, at which time it will ramp down the motion.

T3.SHUTDOWNplans the sequence of steps required to power down the T3 robot and conclude the communication link between it and its host computer.

**Level 2 Primitive - ADT**

INIT turns the power on to the pump, spindle, and ADT, and initializes the ADT.

SHUTDOWN-ADT-PRIM halts the tool motors and servos, and powers down the pump and spindle.

CONTROL: C{P|C}{N|T}{F|P} changes the gain/setpoint for the normal/tangential force/position for the ADT hybrid control laws.

COMMAND - Performs a persistent update of the default parameters. Parameters include: 1) CT sets the orientation of the normal axis relative to the body of the ADT. 2) MH/RB/ES/IS/EM/IM home the ADT motors, reset the bias, and enable or inhibit the servos and motors. 3) ED/IO enable or inhibit the continuous force and position data stream from the ADT. 4) POST-FORCES causes a continuous dump of the ADT force and position information to be captured, filtered, and posted to the WM.

RESET abort the current command or reset the controller with ^C/^P ADT command.

**Level 1 Servo -T3**

MOVE servos to commanded position. Uses T3 interface commands:

SWC causes the T3 to send out its current position.
This is really a request for WM information stored in the proprietary WM of the T3.
This command is sent once after the link has been initialized

RWC causes the T3 to move the commanded coordinates. This command is issued synchronously every 30 ms.

START restarts the T3 controller DDCMP communication link. Message numbers are reset to zero.

**Level 1 Servo -ADT/PLC**

SET - allows persistent setting of the default servo profile parameters (can be changed by the move command)

ADT-ON/ADT-OFF turns on or off the AC power to the ADT.

PUMP-ON/PUMP-OFF turns on or off the AC power to the water pump.

SPINDLE-ON/SPINDLE-OFF turns on or off the spindle power converter.

MOVE - Sets Spindle speed to send an analog voltage signal to the spindle power converter so that the spindle will rotate.

**NAME** GENERIC TEMPLATE **LEVEL** {1 / 2 / 3 / 4 / 5 / 6/ 7}

**OBJECT** Object Hierarchy: System.subsystem.component.subcomponent identifies object domain

**GOAL** The goal can be described as n-vector that defines an attractor value, set point, desired state or as object "to-be" state defined by a map, graph, or geometric definition.

## DESCRIPTION

GENERIC TEMPLATE is an attempt to give the flavor for ascertaining a task functional specification. The description should give a good Natural Language interpretation to the purpose of this task. Many of the following fields in the generic template contain descriptors or functions that may not be necessary in a particular task frame. The purpose of the template is to give a feel for the types of issues that each field is to address.

## PARAMETERS

command name
command number
coordinate frames
<u>object:</u>
as-is descriptor - serial number
to-be descriptor - model number
<u>task related:</u>
projected time to completion
priorities: speed, precision, cost
suggestions: (override defaults)
- best agent fit, best toot use
- tolerance

## STATUS

Control:
echo command name & number
agent and tool list engaged
estimates :
- termination time
- cost

Sensor:
integrated sensor feedback

## AGENTS

The agents describe the set of slaves and properties that the task requires in order to be completed. The properties include the general, kinematic and dynamic characteristics of the agent subsystems

The agents define the subtask slaves commanded by this task.

## REQUIREMENTS

SENSOR FEEDBACK: Requirements include the sensor and world information required during the task. This includes the state of the environement detected by the sensory processing and stored in the world model. This infomraiotn is feedback that conveys actual, or "as-is" conidtions as they exist in the world.

TOOLS: The set of tools, fixtures and jigs that the task requries in order to achieve the goal.

The properties include the general, kinematic and dynamic characteristics of the agent subsystems.

## ENABLING CONDITIONS

The enabling conditions describe the set of assumptions, pre-conditions and initial state that the task makes about the subsystems, environment and self in initiating the task.

## DISABLING CONDITIONS

The disabling conditions describe the set of task exceptions, software and hardware failures that can occur within the task.

## PROCEDURE

The procedural section describes how one achieves the goal state. This includes the plan that describes the subactions, sequence of operations, and agent/tool job strategies. The plan may be described by state graphs, state tables, or in a procedural langauge. The plan will use algorithm, functional, and strategies defined within its level in order to decompose the goal and issue subgoals to its slaves.

NAME | GENERIC_TEMPLATE

## INFORMATION BASE

This section enumerates on the sensorfeedback requirements section.

The World Model Knowledge should {contain | point to,} schemas, and data models

The world knowledge include sections describing state variables, system variables, entity frames, event frames, and maps.

General State and System Variables include:
state clock and sync signals
human interface variables

Level 1) best estimate of time and motion parameters
- ) status of switches or discrete actuators
Level 2) best estimate of time and motion parameters

## PROCEDURAL KNOWLEDGE

BG

The procedural knowledge specifies the routines necessary in control in job assignemnt, planning and state execution. To achieve these goals, the modules typically perorm the following functions (which may assume to exist):

*JOB_ASSIGNER.job_setup*(){
collate_resources(in: work_force, job_parts, out: work_tools, work_agents)
allocate_resources(in: work_agents,work_tools, job_description; out: job_list); --resources available

*PLANNER.plan_generation*(){
schedule(in: job_list, priorities; out:job_schedule); -- optimization of agents, tools, object}
evaluate_alternatives(in: wm.procedures, command.parts, job_schedule(i); out: plan);
interpolate_plan_to_finer_slave_time_coordinates(in: plan; out: plan);}
*PLANNER.contingency or emergency planning*{}

*EXECUTOR.state_interpretation*(){ -- for each slave
wm/sp client interface(in: current_state);
state_evaluator(in:plan, current_state; out:next_state);
logical_to_device-dependent transformations(in:next_state;out:device_specific_state);
send_msg(in: SLAVE(i), device_spcefic_state);
}

SP/WM

This section contains the sensor processing and world model data manipulation and procedural knowledge needs. EXAMPLES:
Client/Service
example_geometric_transformation(in: local_coordinate; out: world_coordinate);

Daemon
*wm.monitor_tools*(in: tool_status_list; out: operator_report);
*wm.monitor_agents*(in: agent_status_list; out: operator_report);
*sp.analyze_performance*(in: job_list_history; out: statistical_analysis);

**NAME** CHAMFER-BATCH **LEVEL** *5-WorkStation*

**OBJECT** CHAMERFING.PART_LIST:<as-is, to-be>

**GOAL** Chamfer each as-is part from the part list into its to-be part within some tolerance.

## DESCRIPTION

**CHAMFER-BATCH** takes an input command that specifies a list of part geometry descriptions with an as-is (serial number) and a to-be (model number) and will chamfer the feature edges of each of the parts.

## PARAMETERS

command name
part list:
- as-is: serial-numbers
- to-be: model-numbers

completion time-frame
priority
- speed
- precision
- cost

## STATUS

Reporting Status:
Percent job Completion
Percent current part Completion
Tool Status(1..i):
-Availability

Agent Status(1..i)
- Availability
- State : Active, On, Off, Error

## AGENTS

CHAMFERING_ROBOT:= T3
- robot w/ chamfering tool

DELIVERY_CART:= NULL

## REQUIREMENTS

TOOLS: CHAMFERING_TOOL:=ADT

VISE
- Jawed Part Gripping

OPERATOR
- Fixtures parts, sends acks

FEEDBACK:
percentage part completion
agent status: running, done, error

## ENABLING CONDITIONS

Assumptions: Appropriate chamfering tool for each part, robot can manipulate chamfering tool with given payload, and dimensionality; vise can handle part(LxWxH) restriction. The material composition is of hard metal: either titanium or inconel. Operator available to deliver and fixture parts when necessary.

## DISABLING CONDITIONS

Exceptions: time-out, subsystem breakdown; power-failure;

## PROCEDURE

```
chamfer_batch(PDES_FILE the_batch)
{   for each part in part_list{
    S1:  plan_generattion();                    // generate or select a plan
    S2:  "instruct operator to fixture this part";   // wait for OK;
    S3:  chamfer_part(the_part);
    S4:  "instruct operator to remove the part"; } }
```

NAME **CHAMFER-BATCH**

## INFORMATION BASE

TASK CONFIGURATION:
State Clock and Sync Signals
Degree Task Completion:
Cost of Task
Task Timing Model
Modes:(off, single step input, single step output, automatic);

GROUP ENTITY FRAMES:
As-is batch
To-be batch

Workstation Entity Frames
- Workstation Coordinate Frame
- Workstation Map (agents,tools, obstacles, etc.)

*Agent Capabilities*: T3, human delivered parts
- availability: dedicated, hence always available
- payload
- envelope: manipulation limits (HxLxW)
- coordinate frame

*Tools*: ADT, PLC, TRAY
Tool Availability
Tool Service Record
coordinate frame

## PROCEDURAL KNOWLEDGE

BG

JOB ASSIGNMENT
*coordinate_resources*(in: work_force, job_parts, out: work_tools, work_agents)
*assign_resources*(in:job_parts; work_tools, work_agents, criteria; out: job_list{part,tools,agents})
-- assign tools/agents to job

PLANNER
*assign_schedule*(in: job_list(part, agents, tools) time_frame, out: job_schedule{part,tool,agent,time))
-- assume part chamfering: 1st come, 1st serve
-- assume tool assignment: 1st available (however since 1 tool is dedicated either busy/available)

EXECUTOR
*configure_subsystem*(in:work_schedule, out: subsystem_configuration);
*monitor_work*(in: work_schedule, wm.job_status; out: send_next_cmd());

SP/WM

Client/Service

*workstation_coordinate_transformations.*{to_agent, to_tool, to_part}

Daemon
*wm.monitor_tools*(in: tool_status_list; out: report);
*wm.monitor_agents*(in: agent_status_list; out: report);
*sp.analyze_performance*(in: job_list_history; out: statistical_analysis);

| NAME | CHAMFER-PART | LEVEL | *4- TASK* |
|---|---|---|---|

**OBJECT** PART_ID

**GOAL** **achieve a process plan for given part that contains a list of feature-derived path points. The process plan should incorporate heuristics to transition the paths along part discontinuities.**

## DESCRIPTION

**CHAMFER-PART** takes a PDES description of a part, knowledge of the finishing requirements, and the location of the part, and plans a sequence of part fixturing and edge chamfering operations for a single part.

| PARAMETERS | STATUS | AGENTS |
|---|---|---|
| command name and number<br>part id -descriptor<br>- as-is part entity frame<br>- to-be part entity frame<br>coordinate system=robot, fixture, tool<br>task priority: speed, precision, cost<br>task duration | echo: command name & number<br>status:=( running, done, error)<br>estimated completion time<br>system margin of error:<br>deviation between expected and sensed | CHAMFERING_ROBOT:= T3<br>- tool attachment<br>-10 mm positioning accuracy<br>- repeatability :5 mm radius |

## REQUIREMENTS

**TOOLS:**

CHAMFERING_TOOL:=ADT

VISE
Jawed Part Gripping
Programmed position control
Operator part placement

**FEEDBACK:**
running, done, error
status:=( running, done, error)
expected feature completion time
deviation between expected and sensed events
-(tooling vs CAD)

## ENABLING CONDITIONS

ASSSUMPTIONS: working T3 and tool set(ADT,PLC) to complete the job stock; part and geometry is available and interpretable.

## DISABLING CONDITIONS

EXCEPTIONS: time-out, tool break, power-failure

## PROCEDURE

```
chamfer_part(part_id)                          // Chamfer part decomposition
{   part analysis(part_id);
        part_features(in: part description; out: feature_list:(fitted curve, force);
        schedule_features(in: job_list, prioritics; out:job_schedule); // optimization of agents, tools, chamfering parts
        process_plan(in:feature_list, out:process_plan)
{   part_machining: with each feature in process_plan
    {   if (feature is not reachable) {
            alert_operator;                    // operator interface command
            operator_ack;                      // feedback await for ok
        }
        chamfer_feature(motion_list.feature(i)); } }// send command to subysstem
}
```

**NAME** CHAMFER-PART

## INFORMATION BASE

TASK CONFIGURATION
State Clock and Sync Signals
Degree Task Completion:
Modes:(off, single step input, single step output, automatic);
*Task Geometry and Clearances*: assumed free space motion

*Workspace Entity Frames and Maps*
Workspace Geometry, Dynamical Model
*Workspace Env. Attributes*: safety only

*Part Entity Frames and Maps*
- Part Geometry - PDES spec. fixturing pose
- Part Manipulation Operations : operator handled
- Part State - un/loaded fixture

RESOURCE CONFIGURATION
- *Agent Entity Frames and Maps*
  -T3: Load kinematic & dynamic model with tool attached
  : Envelope and Payload Limits

- *Tool Entity Frames and Maps*
  - ADT Geometry Offset
  - Vise Location

- *Fixtures Entity Frames and Maps*

## PROCEDURAL KNOWLEDGE

JOB ASSIGNMENT
*coordinate_agents_to_tool* () -- insure 1) tool available, 2) agent-tool matable, 3) tool working and energized
*coordinate_operator_interface_requests():= ok, not ok, retry, stop, abort;*
*allocate_resources*(in: agents, tools, job_description; out: job_list); --resources available

PLANNING
*part_features*(in: part description; out: feature_list:(fitted curve, force){ -- extract all the features of the part
extract_feature_geometry: part interpretation, and curve extraction}
*schedule_features*(in: job_list, priorities; out:job_schedule); -- optimization of agents, tools, chamfering parts
*process_plan*(in:feature_list, out:process_plan) { -- plan "smart" transitions from one feature to the next
BG smoothing_path(in:feature_geometry, chamfering_heuristics; out: process_plan);
*contingency planning*();

EXECUTOR
*ex.feature_smoothing*(in:process_model; out: path_model); -- this applied slave specific upgrades to process plan
*ex.state_execution*(in: path_model, state; out:next_state);

*wm.Part_Geometry_Interpretation*(in:part, out: feature_list{paths,forces})
- assumes PDES data manipulation routines
*wm.chamfering_heuristics*();
- path heuristics model: for smooth transitions along part discontinuities e.g., from curve to edges, corners, etc.
- holistic heuristics model: for optimizing the part fixturing versus chamfering motions (i.e, reduce refixturing,etc)
*wm..Chamfering_Tolerances*(in: part{feature+material}, agent, chamfering_tool; out: acceptable(T/F));
- is the robot & tool accurate enough to maintain the tolerances necessary for chamfering this tool?

SP/WM

| NAME | CHAMFER-FEATURE | LEVEL | *3-EMOVE* |
|---|---|---|---|

**OBJECT** CHAMFER.PART.FEATURE

**GOAL** **derive a series of intermediate (arrival, edge, depart) path points and tool forces given the parameteric representation of the feature**

## DESCRIPTION

**CHAMFER-FEATURE** plans the sequence of robot and tool commands required to approach the feature, activate the tool, track the feature, and depart from the feature. The geometry of the feature is defined in PDES format, the finishing requirements for the feature are also defined, and the position of the part in the vise is known.

| PARAMETERS | STATUS | AGENTS |
|---|---|---|
| command name<br>object A - feature list<br>duration<br>priorities:<br>- speed<br>- accuracy<br>- smoothness<br>criteria:<br>- optimize parts safety<br>- optimize throughput | status: running, done, error<br>expected completion time | ROBOT := T3<br>- position control<br>- |

## REQUIREMENTS

TOOLS:
CHAMFER TOOL := ADT
- chamfering
- chamfering window:= 10 mm radius

**FEEDBACK:**
status: running, done, error
$E_t$( path goal-pt arrival)

## ENABLING CONDITIONS

Assumptions: the chamfering of each part will be determined as a combination of material + tooling force + feedrate.

## DISABLING CONDITIONS

Exceptions: exceeded chamfering forces; unexpected robot singularity.

## PROCEDURE

```
chamfer_feature()
{ S1: ADT->ES(enable); ADT->PLC( pump_on);   ADT->PLC(spindle_on);
  S2: plan motion_list:
      gross_motion(add approach pose to pose list);
      while (there are some vertices left on the feature)
{      fine_motion(add vertex poses of feature to pose list);
       compute and append force/gain parameters for edge on motion_list; }
       gross_motion(add depart pose to motion_list);
  S3: for each feature: T3->move_until(motion_list.item);
      await_signal( CT.wm.event(await signal angle between tool and edge ):);}
```

**NAME** CHAMFER-FEATURE

## INFORMATION BASE

TASK CONFIGURATION
State Clock
Sync Signals : cyclic timing
Modes:(off, single step input, single step output, automatic);
Degree Command Completion:
Objectives: priorities, cost/benefit parameters
Estimated time to goal-pt or contact

Workspace Map & Entity Frame:
- obstacles- vise stand; finished part table
  - geometry & clearances

-Part Entity Frames
- Part Feature Geometry
- Feature Machining Operations - motions, tolerances

RESOURCE CONFIGURATION

-Agent Entity Frame: Geometry, Coordinate System, Motions
T3 Configuration - position, vel, acc limits
T3: Load kinematic & dynamic model with tool attached
T3: Motion Descriptors:
free-space motion, approach-part motion,
depart-part motion, contact motion

- Tool Entity Frame: Geometry and Tolerances
  - ADT Geometry Offset
  - Vise Envelope and Tolerances

## PROCEDURAL KNOWLEDGE

JOB ASSIGNMENT
*job_check*{ PDES_FEATURE: interpretation, resolution, and validation}
*assign_job_priorities*(motion_priorities);
*coordinate_machinery*(); -- insure
PLANNER
*gross_motion_generator*(in: part_feature, agent+tool configuration; out: motion_list)
{ obstacle_avoidance() || canned_motion_intrpretation();
redundancy_resolution();
}*fine_motion_generator*(in: part_feature, agent+tool configuration; out: motion_list)
{ approach_part(); || depart_path(); || initiate_contact(); }
BG *force_field* (in: part_feature, agent+tool configuration; tool_heuristics; out: motion_list)
{ -- compute motion forces and tolerance
compute_forces/gains(in: motion_list; out: motion_list); -- append gains x tangential and normal forces to motion
compute_leeway(in:motion_list; out:motion_list); } -- compute tolerances for error along path
*contingency_planning*();
{ task_exception(in: problem_description; out: problem_prescription){}; replan();}
EXECUTOR
state_interpretation();
logical to device-dependent transformations();

*SP/WM:*
*part_db_interface*()
{ feature_vertices(in: part_feature; out: vertices) || -- determine feature c
feature_composition(in:part_feature; out: force_vector); -- applied forces based on feature }
*Objective_function*(in: priorities, plan; out: process_plan);
*kinematics*(in:t3_configuration; out: model);
*dynamics*(in:t3_configuration; out: model) { jacobian();
SP/WM *coordinate transformation*()
{ part_to_world( in: local_coordinate_frame, type; out: world_coordinate_frame); }
*exception_handlers*(in: problem_description; out: problem_prescription);
- arithmetic and mathematical ill-conditioning exceptions

**NAME** T3->EMOVE.MOVE **LEVEL** *3-EMOVE*

**OBJECT** NULL

**GOAL** **derive a series intermediate obstacle-free path points to achieve the desired position state within given tolerance**

## DESCRIPTION

**T3->MOVE** plans the collision-free path from the current pose of the T3 robot to the given goal pose within the desired completion tolerance and time frame. The task must evaluate the environment elements to avoid collisions. The task will translate from commanded coordinate system into the local (T3) coordinate system.

| PARAMETERS | STATUS | AGENTS |
|---|---|---|
| command name<br>goal-position descriptor<br>motion descriptor (list?)<br>- goal-pt<br>- {series of intermediate pts.}<br>- type: stop pt. vs. fly-through pt.<br>desired duration | status: running, done, error<br>expected completion time | AGENT: MANIPULATION_ROBOT<br>:=WORKSYSTEM.MANIPULATION_ROBOT.T3.;<br>Properties:sensed_force(wrist) |

## REQUIREMENTS

| | |
|---|---|
| TOOLS: CHAMFER TOOL := ADT<br>Properties: chamfering<br>END-EFFECTOR := GRIPPER | SLAVES FEEDBACK<br>status: running, done, error |

| ENABLING CONDITIONS | DISABLING CONDITIONS |
|---|---|
| Assumption: the collision free path is derived from programmed points (not sensed). | Exceptions: |

## PROCEDURE

```
t3-> move(goal-pt)
{ gross_motion_generator(); -- compute_obstacle_free_path
                            -- check for robot singularities and configure robot redundancy
  assign_motion_parameters(in:t3->system; pose_list; out: motion_profile); -- vel, acceleration,etc motion parameters
  robot->move(pose_list{poses; motion_profile});
}
```

NAME | T3->EMOVE.MOVE

## INFORMATION BASE

TASK CONFIGURATION
State Clock
Sync Signals
Modes:(off, single step input, single step output, automatic);
Degree Task Completion:
Objectives: priorities, cost/benefit parameters
Estimated time to goal-pt or contact
Workspace Map & Entity Frames
- obstacles- vise stand; finished part table
  - geometry & clearances

-Part Entity Frame:
- Part Feature Geometry & Location
- Feature Machining Operations - motions, tolerances

RESOURCE CONFIGURATION
- Agent Entity Frame Geometry, Coordinate System, Motions
T3 Configuration - position, vel, acc limits
T3: Load kinematic & dynamic model with tool attached
T3: Motion Descriptors:
free-space motion, approach-part motion,
depart-part motion, contact motion

- Tool Entity Frame: Geometry and Tolerances
  - ADT Geometry Offset
  - Vise Envelope and Tolerances

## PROCEDURAL KNOWLEDGE

JOB ASSIGNMENT
*job_check*{ PDES_FEATURE: interpretation, resolution, and validation}
*assign_job_priorities*(motion_priorities);

PLANNER
*gross_motion_generator*(in: part_feature, agent+tool configuration; out: motion_list{pose,profile})
{ obstacle_avoidance() || canned_motion(); // either use canned paths or avoid obstacles in real-time
  redundancy_resolution();
}

BG *contingency_planning*();
{ wm_job_exception(in: problem_description; out: problem_prescription);
  replan(in:problem_prescription; out: process_plan}

EXECUTOR
*state_interpretation*();
*logical_to_device_dependent_transformations*();

*obstacle_avoidance*(in:workspace+robot+tool configuration, cuurent&goal{pt}; out: path_points); // future implementation
*canned_motion*(in:current&goal{pt}, out: path_points);
*kinematics*(in:t3_configuration; out: model);
*dynamics*(in:t3_configuration; out: model) { jacobian();
*coordinate transformation*()
{ part_to_world( in: local_coordinate_frame, type; out: world_coordinate_frame); }
*exception_handlers*(in: problem_description; out: problem_prescription);
SP/WM - arithmetic and mathematical ill-conditioning exceptions

| NAME | ADT->COMMAND | LEVEL | *2-PRIM* |
|---|---|---|---|

**OBJECT** PART DESCRIPTION

**GOAL** adt->command takes a logical chamfering tool goal state (specified in the parameter list) and translates these goals into the physical device actions

## DESCRIPTION

**ADT->COMMAND** is the logical interface to the ADT Chamfering device. The ADT coordinates the chamfering tool with the PLC to control the various

| PARAMETERS | STATUS | AGENTS |
|---|---|---|
| command name<br>command number<br>as-is: current-state<br>to-be: set/reset/stop<br>force profile:<br>- orientation normal force<br>- normal force limit<br>- tangential force limit<br>- forces: enable/inhibit | echo command name and number<br>status: running, done, error<br>integrated force and feedback data | ADT INTERFACE<br>- ADT servos<br>- ADT motors<br>- ADT command interface |

## REQUIREMENTS

TOOLS:

ADT:
- 1KHz servo rate

WATER PUMP:
- forced water spray

ADT FEEDBACK:
force and position feedback:

## ENABLING CONDITIONS

Assumptions: The ADT serial interface is ready for communication.

## DISABLING CONDITIONS

Exceptions: connection_timeout;

## PROCEDURE

```
procedure ADT_COMMAND(motion_profile);
{   disassemble_parameter_list(in:parameter_list; out: device_specific_list);
    validate_list(in,out:device_specific_list); // insure that order of ADT set/reset operations is coherent
    while(device_specific_list.iitems)
        {   ADT->SET(in: device_specific_list.{command, parameters} }
            await_ack();
        }
}
```

**NAME** ADT->COMMAND

## INFORMATION BASE

| | |
|---|---|
| TASK CONFIGURATION<br>State Clock and Sync Signals<br>Degree Task Completion:<br>Modes:(off, single step input, single step output, automatic);<br><br>*ADT Entity Frame*<br>Force Profile:<br>- orientation of normal axis<br>- tangential force limits<br>- normal force limits<br>- gains<br>- setpoint | Force Feedback:= (disabled, enabled (continuous, sampled);<br><br>*Water Pump Entity Frame*<br>State:=(on,off)<br><br>Part Entity Frame<br>name, type, friction characteristics, stiffness, tolerances |
| Power State: on/off<br>Spindle Speed:= { analog voltage }<br>Motors State:= {enabled, disabled};<br>Servo State:= {enabled, disabled};<br>Lapsed Time since Bias Reset | |

## PROCEDURAL KNOWLEDGE

BG

JOB ASSIGNMENT:
*classify_job*(in:to-be; out:set/reset/stop/pause/resume);

PLANNING:
*decompose_parameters*(in:parameter_list; out:ADT_command_list);

EXECUTOR
*commication_interface*(in: next_state; out: ADT-command);
*await_ack*();

*set*(); - Performs a persistent update of the default parameters.
1) CT sets the orientation of the normal axis relative to the body of the ADT.
2) MH/RB/ES/IS/EM/IM home the ADT motors, reset the bias, and enable or inhibit the servos and motors.
3) ED/IO enable or inhibit the continuous force and position data stream from the ADT.
4) POST-FORCES causes a continuous dump of the ADT force and position information to be captured, filtered, and posted to the WM.
*reset*(); - ^C/^P abort the current command or reset the controller

SP/WM

*translate_rpm_to_voltage(*in: rpm; out: ADT_analog_voltage)();

*verify_setting*(in: part_model, chamfer_model; out: valid(T/F));

| NAME | T3->PRIM.MOVE | LEVEL | 2-PRIM |
|---|---|---|---|
| OBJECT | CHAMFER.LIST.PART.FEATURE | | |
| GOAL | **from a goal position description produce a series of trajectory points using desired algorithm** | | |

## DESCRIPTION

**T3->PRIM.MOVE** takes a list of intermediate poses, and plans a trajectory through those poses. Each intermediate move can be either straight, circular, or a B-spline. Motion between each intermediate pose is smoothed such that the velocity of the tool center point is non-zero between any successive poses, and zero at the end. Parameters for each successive move differ depending upon the nature of the move. Straight moves require that the speed and acceleration be specified or are set to some default. Circular moves require a parameter to specify the circle (such as center point or a third point on the circle). Spline moves require spline

| PARAMETERS | STATUS | AGENTS |
|---|---|---|
| command name<br>command number<br>algorithm<br>coordinate frame of motion<br>:= Cartesian, joint<br>position command description<br>held tool<br>termination condition(s): stop, go-thru<br>redundancy resolution<br>priority<br>objective function<br>trajectory deviation tolerance | echo command name, number<br>running, done, error | ARTICULATED_ROBOT:=<br>- joints |

## REQUIREMENTS

| | |
|---|---|
| TOOLS:<br>NONE | SLAVE FEEDBACK<br>- servo joint positions |

| ENABLING CONDITIONS | DISABLING CONDITIONS |
|---|---|
| Assumptions: path is free of obstacles | Exceptions: exceeded velocity, acceleration, joint limits; |

## PROCEDURE

```
move(motion_profile{pose, dynamic parameters})
{    while (there are some poses)
     {    plan_trajectory;
          execute_trajectory;
     }
```

NAME | T3->PRIM.MOVE

## INFORMATION BASE

| | |
|---|---|
| TASK CONFIGURATION<br>State Clock and Sync Signals<br>Degree Task Completion:<br>Modes:(off, single step input, single step output, automatic);<br><br>RESOURCE CONFIGURATION<br><br>Agent Entity Frame<br>Robot Profile: links, lengths, displacements<br>Forward_Kinematics_model;<br>Inverse_Kinematics_model;<br>Redundancy Configuration;<br><br>Gain DB: not needed | Held Tool Geometry<br>ADT Tool Offset Profile();Lxs<br>configuration |

## PROCEDURAL KNOWLEDGE

BG

Job Assignment
*fetch_pose_description*(pose id); -- get pose id from wm db

Planner
*Straight_line_interpolation*(end-points, tool_offset); -- given path, produce a set of joint path points
*Circular_path_interpolations*(end-points, tool_offset); -- given path, produce a set of joint path points
*Quintic_polynomial_path_interpolations*(end-points, tool_offset); -- given path, produce a set of joint path points
*B_spline_path_interpolations*(end-points, tool_offset); -- given path, produce a set of joint path points

Executor
*Send_Command*(command_name:=MOVE, desired_joint values);
*Cyclic_ex_process*(in: state; out: next_state);
*communication_interface*(in:motion_list; out: t3_commands); // T3 expects Cartesian or joint commands

SP/WM

| NAME | PORIM->T3.MOVE-UNTIL | LEVEL | 2- T3 PRIM |
|---|---|---|---|

**OBJECT** NULL

**GOAL** attemnpt a goal-position through a series of trajectories until a sensed chamfering force is exceeded

## DESCRIPTION

MOVE-UNTIL takes a single pose, and will move in either a straight, circular, or B-spline path until a prescribed force is attained, at which time it will ramp down the motion.

| PARAMETERS | STATUS | AGENTS |
|---|---|---|
| command name<br>command number<br>EE force limit<br>algorithm<br>coordinate frame of motion<br>:= Cartesian, joint<br>position command description<br>held tool<br>termination condition(s): stop, go-thru<br>redundancy resolution<br>priority<br>objective function<br>trajectory deviation | echo command name, number<br>running, done, error | ARTICULATED_ROBOT:=<br>- joints(1..6) |

## REQUIREMENTS

| | |
|---|---|
| | SLAVE FEEDBACK<br>- servo joint positions<br>- ADT force readings |

## ENABLING CONDITIONS

Assumptions: path is free of obstacles and ADT End-Effector registers force limits.

## DISABLING CONDITIONS

Exceptions: exceeded velocity, acceleration, joint limits.

## PROCEDURE

```
move_until(motion_profile)
{   post_forces(on); /*this is an ADT PRIM command */
    while(there are some poses) {                    plan_trajectory;
        {spawn_event(if forces exceeded, break any robot motion);
         halt(if forces exceeded, break any robot motion);
        }
        execute_trajectory; }

    post_forces(off); }
```

**NAME** PRIM->T3.MOVE-UNTIL

## INFORMATION BASE

TASK CONFIGURATION
State Clock and Sync Signals
Degree Task Completion:
Modes:(off, single step input, single step output, automatic);

Part Entity Frame
name, type, friction characteristics, stiffness, tolerances

RESOURCE CONFIGURATION

Agent Entity Frame
Robot Profile: links, lengths, displacements
Kinematic Profile;

RESOURCE GEOMETRY
ADT WM Interface
observed normal and tangential forces

Held Tool Geometry
ADT Tool Offset Profile();Lxs
configuration

## PROCEDURAL KNOWLEDGE

BG

Job Assignment
Fetch_Part_Description(part id); -- get part id from
WM:ADT:post_forces(on); -- (prim talking to prim through world model?)

Planner
Straight_line_interpolation(end-points, tool_offset); -- given path, produce a set of joint path points
Force_threshold(part); -- determine force threshold, direction, tolerance

Executor
*ADT->read_forces*();
*Send_Command*(command_name:=MOVE, desired_joint values);

SP/WM

*forward_kinematics*();
*inverse_kinematics*();

*exceed_force_limit_event*(in: ADT->chamfering_force, force_limit; out: event_wakeup);

| NAME | T3->SERVO | LEVEL | *1: T3- SERVO* |
|---|---|---|---|

**OBJECT** NULL

**GOAL** **decompose a tractory point into a actuator command.**

**DESCRIPTION**

MOVE - move robot to desired goal position specified in accordinace with pose profile. Decompose such a trajectory point into a series of actuator/servo commands.

| PARAMETERS | STATUS | AGENTS |
|---|---|---|
| command name<br>command number<br>coordinate system: joint,Cartesian<br>desired position<br>desired velocity<br>desired acceleration<br>servo loop gains<br>servo algorithm:= PD, PID | echo command name, number<br>running, done, error<br>current joint positions | SERVOS := T3_maninpulator links(1..6) Cartesian or joint position control<br>Properties:= (Communication,assumes an interface to the robot joint servo controller) |

**REQUIREMENTS**

| | |
|---|---|
| | FEEDBACK<br>- joint/actuator (1..6) reading // actual readings<br>- |

| ENABLING CONDITIONS | DISABLING CONDITIONS |
|---|---|
| Assumptions: the communication link to the robot is powered and | Exceptions: exceed hardware joint reach, velocity, acceleration limits; |

**PROCEDURE**

```
servo(){
sp.get_current_position(out:tick_joints);                    // sp feedback and wm
wm.conversion(in:tick_joints, out:current_joints);
compute_interpolated_path(in: current, desired, path(i..i): joints, time_quantum); //planning function
for i quantum ticks :                                        // execution
get_current_position(current_joints);
compute_next(current_position, desired_position, time_period);
communicate_desired_position(desired_position(1..i));
```

NAME: T3->SERVO

## INFORMATION BASE

TASK CONFIGURATION
State Clock and Sync Signals
Modes:(off, single step input, single step output, automatic);

RESOURCE CONFIGURATION
Agent Entity Frame
Observed state vector defining pose, velocity, acceleration
Jacobian matrix (not needed)
- inverse kinematic coordiante transform(tool => actuator)
Servo Update frequency: ionterval
Robot update frequency: interval time
Robot links: lengths, displacements, ticks

T3 Communication
- Message Format
- Command Interface

## PROCEDURAL KNOWLEDGE

BG

Job Assignment
*system_checkout*() -- insure system on and responding

Planner
*time_interpolate*(in: joint_list; out: joint_interp_list) -- interpolate point over time interval

Executor
*rwc*(out: joints) -- send desired joints to robot
*swc* (in: tick_joints) -- get current joints from robot

*link_control*(in: type) -- start or acknolwedge link to robot

SP/WM

*forward_kinematics*();
*inverse_kinematics*();
*coordinated_joint_motion*();
*conversion* (in : joint) -- converts current robot joint ticks to angular joints

| NAME | SET_PUMP/ADT/SPINDLE | LEVEL | *1: PLC SERVO* |
|---|---|---|---|

OBJECT: NULL

GOAL:

**DESCRIPTION**

SET-PUMP set or reset bit in digital I/O port to control The object is to take a logical command and translate it into a device specific command.

| PARAMETERS | STATUS | AGENTS |
|---|---|---|
| command name<br>attribute:<br>set_pump: on/off<br>set_adt: on/off<br>set_spindle: on/off<br>set spindle_speed:<br>(int) velocity in ?/sec | | PLC Interface<br>Properties:= (Communication,(RS-232 serial communication link to PLC, 9600 baud, 8 bit, 1 stop, no parity)) |

**REQUIREMENTS**

STATUS FEEDBACK
status: running, done, error

SLAVE FEEDBACK
Communication Feedback: ack

**ENABLING CONDITIONS**

**DISABLING CONDITIONS**

**PROCEDURE**

NAME | SET_PUMP/ADT SPINDLE

## INFORMATION BASE

## PROCEDURAL KNOWLEDGE

BG

JOB ASSIGNMENT
dveice_configure(); -- insure device on and ready
command_validate(in:); -- check for legal command
PLANNER
assemble_job_list(in: command; out: device_action_list);

EXECUTOR
*execute_state_change*(){
execute_device_action_list(i:=1,n):
  pump_on(void) { set or reset bit in digital I/O port; } ||
  pump_off(void) { set or reset bit in digital I/O port; } ||
  adt_on(void) { set or reset bit in digital I/O port; } ||+
  adt_off(void) { set or reset bit in digital I/O port; } ||
  spindle_on(void) { set or reset bit in digital I/O port; } ||
  spindle_off(void) { set or reset bit in digital I/O port; } ||
  spindle_speed(double rpm) { write byte value in analog I/O port; }
await_(PLC, plc_feedback);
interpret_feedback(in: plc_feedback, out: exception | continue_};

SP/WM

# ADACS LEXICON for each Level of the NASREM Hierarchy

## Level 5 Workstation - INPUT

*ex.configure_subsystem*(in:work_schedule, out: subsystem_configuration);

*ex.monitor_work*(in: work_schedule, wm.job_status; out: send_next_cmd());
The execution monitor is responsible for

*ja coordinate_resources*(in: work_force, job_parts, out: work_tools, work_agents)

*ja.assign_resources*(in:job_parts; work_tools, work_agents, criteria; out: job_list{part,tools,agents})
-- assign tools/agents to job

*pl.assign_schedule*(in: job_list(part, agents, tools) time_frame, out: job_schedule{part,tool,agent,time))
-- assume part chamfering: 1st come, 1st serve
-- assume tool assignment: 1st available (however since 1 tool is dedicated either busy/available)

*wm.monitor_tools*(in: tool_status_list; out: report);
*wm.monitor_agents*(in: agent_status_list; out: report);
*sp.analyze_performance*(in: job_list_history; out: statistical_analysis);

## Level 4 Task

*ja.coordinate_agents_to_tool* () -- insure 1) tool available, 2) agent-tool matable, 3) tool working and energized
*ja.coordinate_operator_interface_requests():= ok, not ok, retry, stop, abort;*
*ja.allocate_resources*(in: agents, tools, job_description; out: job_list); --resources available

*pl.part_features*(in: part description; out: feature_list:(fitted curve, force);
-- extract all the features of the part
*pl.schedule_features*(in: job_list, priorities; out:job_schedule);
-- optimization of agents, tools, chamfering parts
*process_plan*(in:feature_list, out:process_plan);
-- plan "smart" transitions from one feature to the next

*pl. contingency planning*();

*ex.feature_smoothing*(in:process_model; out: path_model); -- this applied slave specific upgrades to process plan
*ex.state_execution*(in: path_model, state; out:next_state);

*wm.Part_Geometry_Interpretation*(in:part, out: feature_list{paths,forces})
- assumes PDES data manipulation routines
*wm.chamfering_heuristics*();
- path heuristics model: for smooth transitions along part discontinuities e.g., from curve to edges, corners, etc.
- holistic heuristics model: for optimizing the part fixturing versus chamfering motions (i.e, reduce refixturing,etc)
*wm.Chamfering_Tolerances*(in: part{feature+material}, agent, chamfering_tool; out: acceptable(T/F));
- is the robot & tool accurate enough to maintain the tolerances necessary for chamfering this tool?

## Level 3 Elemental Move

JOB ASSIGNMENT
*ja.job_check*{ PDES_FEATURE: interpretation, resolution, and validation}
*ja. assign_job_priorities*(motion_priorities);
*ja. coordinate_machinery*(); -- insure machinery available and working

*pl.gross_motion_generator*(in: part_feature, agent+tool configuration; out: motion_list{pose,profile})
The gross motion generator must handle obstacle_avoidance, by using either using canned paths or avoiding obstacles in real-time. The gross motion generator should also handle any kinematic redundancy problems within any subagents.

*pl. contingency_planning*();
Handle job exceptions that arise within the task. The contingency planner should take a problem description and if available prescribe a problem prescription.The contingency planning should take a prescription and replan the process plan based on the alternative strategies and recommendations. If no prescription available, then raise an exception to a higher supervision.

*pl. fine_motion_generator*(in: part_feature, agent+tool configuration; out: motion_list)

*pl. force_field* (in: part_feature, agent+tool configuration; tool_heuristics; out: motion_list);
-- compute motion forces and tolerance

*ex.state_interpretation*();
*ex.logical to device-dependent transformations*();

*wm. part_db_interface*()
-- determine feature and applied forces based on feature
*wm.Objective_function*(in: priorities, plan; out: process_plan);

*wm.kinematics*(in:t3_configuration; out: model);
*wm.dynamics*(in:t3_configuration; out: model);
*wm.coordinate transformation*()
*wm.exception_handlers*(in: problem_description; out: problem_prescription);
-- arithmetic and mathematical ill-conditioning exceptions

*wm.obstacle_avoidance*(in:workspace+robot+tool configuration, cuurent&goal{pt}; out: path_points);

*wm.canned_motion*(in:current&goal{pt}, out: path_points);

**Level 2 Primitive**

*fetch_pose_description*(pose id);
-- get pose id from wm db

*pl.Straight_line_interpolation*(end-points, tool_offset);
-- given path parameters, produce a set of joint path points
*pl.Circular_path_interpolations*(parameters, tool_offset);
-- given path parameters, produce a set of joint path points
*pl.Quintic_polynomial_path_interpolations*(end-points, tool_offset);
-- given path, produce a set of joint path points
*pl.B_spline_path_interpolations*(end-points, tool_offset);
-- given path, produce a set of joint path points
*pl.force_threshold*(part);
-- determine force threshold, direction, tolerance

*ex.Send_Command*(command_name:=MOVE, desired_joint values);
*ex.Cyclic_ex_process*(in: state; out: next_state);
*ex.T3communication_interface*(in:motion_list; out: t3_commands);
-- T3 expects Cartesian or joint commands
*wm.forward_kinematics*();
*wm.inverse_kinematics*();
*wm.exceed_force_limit_event*(in: ADT->chamfering_force, force_limit; out: event_wakeup);

**Level 1 Servo**

*ja.system_checkout*() -- insure system on and responding

*pl.time_interpolate*(in: joint_list; out: joint_interp_list) -- interpolate point over time interval

*ex.communication()*;
-- send desired joints to robot; for T3 *ex.rwc*(out: joints) --
-- get current joints from robot; for T3: *ex.swc* (in: tick_joints) --
-- start or acknowledge link to robot: for T3: *link_control*(in: type) --

# Appendix E: EXPRESS Schemas for ADACS

(*
EXPRESS SCHEMA FOR ADACS TASKS AND GEOMETRY
T. Kramer
revised January 30, 1992

This file contains four schemas needed for ADACS (Advanced Deburring And Chamfering System). All four schemas use EXPRESS version N14 of April 1991. All four schemas pass through the FedEx parser (January 28, 1992 version) without error.

1. adacs_actions - The nature of the adacs_actions schema is described where the schema starts.

2. adacs_geometry - This schema contains entities and types used for describing geometric things like chamfer_faces and robot trajectories. This schema builds on the STEP geometry and topology schemas. The purpose of this schema is to support the adacs_actions schema.

3. geometry - This is a portion of the STEP geometry schema, version N87 of June 1991. It defines entities needed in the adacs_geometry schema. "Where" rules and "derive" clauses have been deleted.

4. topology - This is a portion of the STEP topology schema, version N87 of June 1991. It defines entities needed in the adacs_geometry schema. "Where" rules and "derive" clauses have been deleted.

Each schema may have three parts: (1) interface clauses, (2) type definitions, and (3) entity definitions. The first two are not always present. The type and entity definition sections are arranged alphabetically.

This file is intended to be parsable, so it is full of EXPRESS comment characters.

As compared with the December 18, 1991 version of these schemas, there are two major substantive change, one major format change, and several minor additions and changes.

First Major Substantive Change - The ALPS schema has been deleted. It has been decided in the initial implementation of the ADACS control system to use sequentially executed process plans, so that the complicated facilities of ALPS for handling control flow are not needed. A single entity (adacs_plan) and a type (nasrem_level) have been added to replace ALPS. Instead of using the "PrimitiveTaskNode" of ALPS as the top of the hierarchy of ADACS tasks, a new entity, "adacs_action" has been added at the top of the adacs_actions schema. After the initial implementation of the ADACS control system is complete, the question of whether a high-level process plan language (such as ALPS) is needed will be reconsidered. It is intended that plans will be written in STEP physical files. The currently available STEP tools all use the "Tokyo" version of the specification for STEP physical files, so that version will be used.

Second Major Substantive Change - The chamfer_path entity and all its

subtypes have been deleted, and the functionality of these entities has been incorporated in the chamfer_face entities and its subtypes.

Major Format Change - Entities and types used for describing geometric things like chamfer faces and robot trajectories have been taken out of the ADACS schema and put into a separate adacs_geometry schema.

Other Changes -

A. Added tasks
 1. ADACS E-move level: move_t3
 2. ADACS Servo level: adt_set_speed, t3_servo, t3_control_on, t3_control_off, t3_power_on, t3_power_off, t3_move_to

B. The format for names has been changed so it is not case-sensitive. Instead of using capital letters to separate the parts of a name, underscores are used. Only lower case letters are used here.

C. Supertype statements have been added to all schemas to make it easy to trace the hierarchies downwards, except where a supertype is in a separate schema.

*)

```
(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)
(* START ADACS_ACTIONS SCHEMA *)
```

(*

Only definitions of tasks and definitions of the non-geometric things needed to support the tasks (such as tools and workpieces) are included here. Geometric things that are used in this schema are included in the adacs_geometry schema.

The tasks defined here are intended to be useable both in process plans and in commands given to ADACS controllers to carry out process plans. In order to serve both uses, some tasks have attributes (such as the identity of a specific workpiece) that are optional because they are known at execution time but not at planning time.

The top-level node type for ADACS tasks is adacs_action, which has four subtypes, one for each control level covered by the schema.

It is implicit in this schema that the ADACS workstation will include one robot and one ADT and that the tool in the ADT will not be changed during a single chamfer_part operation.

The ADT is a single physical device that corresponds to two logical devices in the NASREM model: a sensory system and an agent that executes tasks. This schema deals only with tasks for agents that execute tasks. Thus, no tasks for getting data from the ADT have been included here, even though commands must be given to the ADT to get

data from it. It is assumed that the ADACS controllers will share the physical communication link to the ADT with the ADACS sensory processing and world modeling (SPWM) module(s) that need to communicate with the ADT, and the SPWM modules will deal with the ADT when data is needed. If taking turns giving commands to the ADT proves unworkable or clumsy, then additional definitions may be added to this schema for getting data from the ADT, making getting data an executable task.

It has been decided to have initialization and shutdown be accomplished throughout ADACS by commands issued by the controllers. Thus, this schema contains a lot of commands for this purpose. The alternative would be to treat initialization and shutdown outside of the normal command system. If a switch is made to this alternative, many commands defined here could be deleted.

The entities in this schema consist of a hierarchy of adacs_actions plus three other entities (adacs_plan, chamfer_list, and design_id) that are not adacs_actions.

The hierarchy of adacs_actions is as follows:

```
adacs_action
      adacs_e_move
            adacs_e_move_shutdown
            adacs_e_move_startup
            chamfer_feature
            fixture_part
            unfixture_part
      adacs_primitive
            adt_primitive
                  adt_apply_force
                        adt_shutdown
                        adt_startup
            t3_primitive
                  t3_move
                        t3_move_until
                  t3_shutdown
                  t3_startup
      adacs_task
            adacs_task_shutdown
            adacs_task_startup
            chamfer_part
      adacs_servo
            adt_servo
                  adt_control_off
                  adt_control_on
                  adt_power_off
                  adt_power_on
                  adt_pump_off
                  adt_pump_on
                  adt_set_coordinate_rotation
                  adt_set_gains
                  adt_set_setpoint
```

```
                adt_set_speed
                adt_spindle_off
                adt_spindle_on
          t3_servo
                t3_control_off
                t3_control_on
                t3_move_to
                t3_power_off
                t3_power_on

*)

SCHEMA adacs_actions;

(*****************************************************************************)
(*****************************************************************************)
(* START ADACS_ACTIONS INTERFACE CLAUSES *)

USE FROM adacs_geometry (chamfer_face, pose, trajectory);

(* END ADACS_ACTIONS INTERFACE CLAUSES *)
(*****************************************************************************)
(*****************************************************************************)
(* START ADACS_ACTIONS TYPE DEFINITIONS *)

(* chamfer_instruction  ***************************************************

A chamfer_instruction is either a chamfer_string (a TYPE) or
a chamfer_list (an ENTITY).

*)

TYPE chamfer_instruction = SELECT (chamfer_string, chamfer_list);
END_TYPE;

(* chamfer_string  ********************************************************

This is a string describing in English what sort of chamfers to make.
It is not intended to be machine-usable, so there are no syntax or
grammar requirements.

Example: "Chamfer all hole edges half a millimeter"

*)

TYPE chamfer_string = STRING;
END_TYPE;

(* fixture_id  ************************************************************

This identifies a specific physical object which is a fixture. It is like
an inventory number.

*)
```

```
TYPE fixture_id = STRING;
END_TYPE;

(* nasrem_level ************************************************************

This identifies the hierarchical level of a plan. The choice is one of
the six standard NASREM levels.

*)

TYPE nasrem_level = ENUMERATION OF
                   (cell,
                    workstation,
                    task,
                    e_move,
                    primitive,
                    servo);
END_TYPE;

(* plan_id *****************************************************************

This identifies a process plan.

*)

TYPE  plan_id = STRING;
END_TYPE;

(* tool_id  ****************************************************************

This identifies a specific physical object which is a tool. It is like
an inventory number.

*)

TYPE tool_id = STRING;
END_TYPE;

(* tool_type_id  ***********************************************************

This identifies a type of tool. It is like a catalog number.

*)

TYPE tool_type_id = STRING;
END_TYPE;

(* workpiece_id  ***********************************************************

This identifies a specific physical object.

*)

TYPE workpiece_id = STRING;
END_TYPE;
```

```
(* END ADACS_ACTIONS TYPE DEFINITIONS *)
(*****************************************************************************)
(*****************************************************************************)
(* START ADACS_ACTIONS ENTITY DEFINITIONS *)


(* adacs_action ************************************************************

This is the top level entity for ADACS tasks. An adacs_action may be a
step in a process plan or it may be a command. This entity serves as
the supertype of tasks at four specific control levels.

It has only one attribute, a sequence number. It is intended that
sequence numbers be assigned to plan steps or commands in numerical
order (starting with 1) and that they be executed in this order.

It may be desirable to add more attributes.

*)

ENTITY adacs_action
  SUPERTYPE OF (ONEOF
                   (adacs_e_move,
                    adacs_primitive,
                    adacs_task,
                    adacs_servo));
      sequence_number: INTEGER;
END_ENTITY;

(* adacs_e_move ************************************************************

This is the ADACS top-level node type for the E-Move level. It serves
only as a single supertype for all E-Move level level nodes and has no
attributes.

*)

ENTITY adacs_e_move
  SUPERTYPE OF (ONEOF
                   (adacs_e_move_shutdown,
                    adacs_e_move_startup,
                    chamfer_feature,
                    fixture_part,
                    unfixture_part))
  SUBTYPE OF (adacs_action);
END_ENTITY;

(* adacs_e_move_shutdown ***************************************************

This shuts down the E-move level controller. After this command is
received, the only command that may be given is a startup. Any other
command will be ignored. More typically, of course, the shutdown
command will be followed by powering down the system. When the
shutdown command is received, Primitive level shutdown commands will
```

```
be given to the ADT and T3 Robot.

*)

ENTITY adacs_e_move_shutdown SUBTYPE OF (adacs_e_move);
END_ENTITY;

(* adacs_e_move_startup ***************************************************

This starts up the E-move level controller so it is ready to accept
other commands. After the controller is powered up and reaches a state
where it can read commands, any other command will be ignored until
this command is given. When this command is received, Primitive level
startup commands will be given to the ADT and T3 Robot.

*)

ENTITY adacs_e_move_startup SUBTYPE OF (adacs_e_move);
END_ENTITY;

(* adacs_plan *************************************************************

This is the entity which collects adacs_actions together into a plan, in
case it is desired to formulate an entire plan. It is implicit here that
all the steps in the plan will be executed sequentially in the order
given by the list of steps.

All the steps must be subtypes of the type appropriate for the given
control_level. If control level is task, all steps must be subtypes of
adacs_task. If control level is e_move, all steps must be subtypes of
adacs_e_move. Etc.

The string which is the header should contain information such as
who wrote the plan, when it was written, what version this is, etc.
It may be desirable to formalize more detail in the header, eventually.

*)

ENTITY adacs_plan;
      identity:   plan_id;
      header:           STRING;
      control_level:    nasrem_level;
      steps:            LIST [1 : ?] OF adacs_action;
END_ENTITY;

(* adacs_primitive ********************************************************

This is the ADACS top-level node type for the Primitive level. It
serves only as a single supertype for all Primitive level nodes and
has no attributes.

*)

ENTITY adacs_primitive
  SUPERTYPE OF (ONEOF
```

```
                    (adt_primitive,
                     t3_primitive))
  SUBTYPE OF (adacs_action);
END_ENTITY;

(* adacs_servo ************************************************************

This is the ADACS top-level node type for the Servo level. It serves
only as a single supertype for all Servo level nodes and has no
attributes.

*)

ENTITY adacs_servo
  SUPERTYPE OF (ONEOF
                    (adt_servo,
                     t3_servo))
  SUBTYPE OF (adacs_action);
END_ENTITY;

(* adacs_task *************************************************************

This is the ADACS top-level node type for the task level. It serves
only as a single supertype for all task level nodes and has no
attributes.

*)

ENTITY adacs_task
  SUPERTYPE OF (ONEOF
                    (adacs_task_shutdown,
                     adacs_task_startup,
                     chamfer_part))
  SUBTYPE OF (adacs_action);
END_ENTITY;

(* adacs_task_shutdown ****************************************************

This shuts down the Task level controller. After this command is
received, the only command that may be given is a startup. Any other
command will be ignored. More typically, of course, the shutdown
command will be followed by powering down the system. When the
shutdown command is received, an E-move level shutdown command will
be given to the E-move controller.

*)

ENTITY adacs_task_shutdown SUBTYPE OF (adacs_task);
END_ENTITY;

(* adacs_task_startup *****************************************************

This starts up the Task level controller so it is ready to accept
other commands. After the controller is powered up and reaches a state
where it can read commands, any other command will be ignored until
```

```
this command is given. When this command is received, an E-move level
startup command will be given to the E-move controller.

*)

ENTITY adacs_task_startup SUBTYPE OF (adacs_task);
END_ENTITY;

(* adt_apply_force ***********************************************************

It is anticipated here that the form of the control law for the ADT
will not be changed (although values of the parameters in the law may
be reset), so that the law itself does not need to be communicated.

It is also assumed that the ADT will always be turned on with coolant
flowing and the spindle turning when it is used, so that these
conditions do not need to be specified at the Primitive level.

*)

ENTITY adt_apply_force SUBTYPE OF (adt_primitive);
        angle:                  INTEGER;
        normal_force_gain:              INTEGER;
        normal_position_gain:           INTEGER;
        tangential_force_gain:          INTEGER;
        tangential_position_gain:       INTEGER;
        normal_force:               INTEGER;
        normal_position:            INTEGER;
        tangential_force:           INTEGER;
        tangential_position:        INTEGER;
      spindle_speed:                INTEGER;
WHERE
        0                         <= normal_force_gain;
        normal_force_gain         <= 3000;
        0                         <= normal_position_gain;
        normal_position_gain      <= 1000;
        0                         <= tangential_force_gain;
        tangential_force_gain     <= 3000;
        0                         <= tangential_position_gain;
        tangential_position_gain  <= 1000;
        -4000                     <= normal_force;
        normal_force              <= 4000;
        -1150                     <= normal_position;
        normal_position           <= 1150;
        -4000                     <= tangential_force;
        tangential_force          <= 4000;
        -1150                     <= tangential_position;
        tangential_position       <= 1150;
      0                         <= spindle_speed;
      spindle_speed             <= 90000;
END_ENTITY;

(* adt_control_off ***********************************************************

This shuts down the application of the ADT control law.
```

```
*)

ENTITY adt_control_off SUBTYPE OF (adt_servo);
END_ENTITY;

(* adt_control_on ************************************************************

This starts application of the ADT control law. If the gains and set
point have not been explicitly set, they take on default values when
this is issued. All four gains default to 1000. The other six parameters
from adt_apply_force all default to 0.

*)

ENTITY adt_control_on SUBTYPE OF (adt_servo);
END_ENTITY;

(* adt_power_off *************************************************************

This turns off power to the ADT. The spindle will not keep turning
and the ADT will not follow its control law when the power is off.

This command does not affect the pump.

*)

ENTITY adt_power_off SUBTYPE OF (adt_servo);
END_ENTITY;

(* adt_power_on  *************************************************************

This turns on power to the ADT. The spindle cannot be turned on and
the ADT cannot be enabled unless the power is on, but turning on the
power neither starts the spindle nor enables the ADT.

*)

ENTITY adt_power_on SUBTYPE OF (adt_servo);
END_ENTITY;

(* adt_primitive *************************************************************

This is a subtype of adacs_primitive to be directed to the ADT.

*)

ENTITY adt_primitive
  SUPERTYPE OF (ONEOF
                  (adt_apply_force,
                   adt_shutdown,
                   adt_startup))
  SUBTYPE OF (adacs_primitive);
END_ENTITY;
```

```
(* adt_pump_off ************************************************************

Turns off water pump for the ADT. This is independent of whether ADT
power is on or off.

*)

ENTITY adt_pump_off SUBTYPE OF (adt_servo);
END_ENTITY;

(* adt_pump_on *************************************************************

This turns on the water pump for the ADT. This is independent of
whether ADT power is on or off.

*)

ENTITY adt_pump_on SUBTYPE OF (adt_servo);
END_ENTITY;

(* adt_servo ***************************************************************

A suite of ADT Servo commands is defined in this schema to match the
commands that can actually be given to the ADT. They are all subtypes of
this adt_servo command. In two cases, a single command here would
break down into four commands for the ADT.

There are no ADT Servo commands here corresponding to "RB" (reset bias) or
"MH" (motor home). If it is planned to use these, commands will be added
to this schema.

As discussed at the beginning of this schema, there are also no ADT
Servo commands for getting data from the ADT.

*)

ENTITY adt_servo
  SUPERTYPE OF (ONEOF
                   (adt_control_off,
                    adt_control_on,
                    adt_power_off,
                    adt_power_on,
                    adt_pump_off,
                    adt_pump_on,
                    adt_set_coordinate_rotation,
                    adt_set_gains,
                    adt_set_setpoint,
                    adt_set_speed,
                    adt_spindle_off,
                    adt_spindle_on))
 SUBTYPE OF (adacs_servo);
END_ENTITY;

(* adt_set_coordinate_rotation *********************************************
```

```
This rotates the coordinate system in which the ADT setpoint is
expressed and forces are measured by the given angle (in degrees) from
its default position with respect to the body of the ADT. The angle
may be any integer, positive or negative.

*)

ENTITY adt_set_coordinate_rotation SUBTYPE OF (adt_servo);
      angle:      INTEGER;
END_ENTITY;

(* adt_set_gains ************************************************************

This sets the four gains in the ADT control law. The form of the law is
fixed.

*)

ENTITY adt_set_gains SUBTYPE OF (adt_servo);
      normal_force_gain:            INTEGER;
      normal_position_gain:         INTEGER;
      tangential_force_gain:        INTEGER;
      tangential_position_gain:     INTEGER;
WHERE
      0                         <= normal_force_gain;
      normal_force_gain         <= 3000;
      0                         <= normal_position_gain;
      normal_position_gain      <= 1000;
      0                         <= tangential_force_gain;
      tangential_force_gain     <= 3000;
      0                         <= tangential_position_gain;
      tangential_position_gain  <= 1000;
END_ENTITY;

(* adt_set_setpoint *********************************************************

This sets the ADT setpoint in terms of the current coordinate system.
The units for positions are thousandths of centimeters.
The units for force are millinewtons.

*)

ENTITY adt_set_setpoint SUBTYPE OF (adt_servo);
      normal_force:             INTEGER;
        normal_position:          INTEGER;
        tangential_force:         INTEGER;
        tangential_position:      INTEGER;
WHERE
        -4000               <= normal_force;
        normal_force        <= 4000;
        -1150               <= normal_position;
        normal_position     <= 1150;
        -4000               <= tangential_force;
        tangential_force    <= 4000;
        -1150               <= tangential_position;
```

```
        tangential_position <= 1150;
END_ENTITY;

(* adt_set_speed ************************************************************

This sets the spindle speed (in rpm) at which the ADT will turn when it
is on.

*)

ENTITY adt_set_speed SUBTYPE OF (adt_servo);
      speed:      INTEGER;
WHERE
      0 <= speed;
        speed <= 90000;
END_ENTITY;

(* adt_shutdown *************************************************************

This is a Primitive level command for shutting down the ADT. When this
command is received, the Primitive level ADT controller sends adt_spindle_off,
adt_control_off, adt_power_off, and adt_pump_off commands to the ADT Servo
level controller.

*)

ENTITY adt_shutdown SUBTYPE OF (adt_primitive);
END_ENTITY;

(* adt_spindle_off **********************************************************

This turns off the spindle.

*)

ENTITY adt_spindle_off SUBTYPE OF (adt_servo);
END_ENTITY;

(* adt_spindle_on ***********************************************************

If the ADT power is on, this turns on the spindle. It will turn at the
rate that is set by adt_set_speed. If no adt_set_speed command has been
given, the speed defaults to zero.

*)

ENTITY adt_spindle_on SUBTYPE OF (adt_servo);
END_ENTITY;

(* adt_startup **************************************************************

This is a Primitive level command for starting up the ADT. When this
command is received, the Primitive level ADT controller sends
adt_pump_on, adt_power_on, and adt_control_on commands to the ADT Servo
level controller. The adt_spindle_on command is not given.
```

```
*)

ENTITY adt_startup SUBTYPE OF (adt_primitive);
END_ENTITY;


(* chamfer_feature ***********************************************************

This produces one chamfer_face.

This does not require identification of a tool or workpiece, since that
has been done in chamfer_part.

chamfer_to_make - Referencing a chamfer_face (as defined in the
adacs_geometry schema) seems preferable to referencing a STEP feature
since the chamfer_face provides a sufficient description of the
chamfer to be produced. The STEP edge_flat feature from the Form
Features Information Model comes close to being usable but does not
have several of the desired characteristics.

spindle_speed - It is assumed that spindle speed for the ADT will be
constant throughout a chamfer_feature operation. The value of the
speed is given in rpm by this attribute.

chamfer_feature_plan_id - If a process plan has already been made giving
a series of tasks for the Primitive level that will produce the
chamfer_face, this identifies it. If the controller executing a
chamfer_feature command is expected to determine tasks for subordinates
in real time, this attribute has no value.

The scenario assumed here for chamfer_feature is that several passes
may be required to make a single chamfer_face. In each pass, (1) the
ADT will have its set point set so it will not contact the part at the
start of the pass (2) the robot will move the ADT to the robot start
point for the pass, (3) the ADT set point will be changed to bring the
tool into contact with the part, (4) the robot will move the ADT
through a trajectory (series of poses) to do some chamfering, (5) the
robot will move the ADT away from the part.

Variations on the five items are possible. It may turn out to be
better to use the robot to bring the ADT into contact with the part
than to have the ADT move itself. It is not expected that resetting
ADT parameters will be done during the portion of a pass in which the
ADT is in contact with the part (step 4 above). If other techniques do
not work, this may be tried.

chamfer_feature may be thought of as decomposing into either a series
of passes (each of which is comprised of a series of ADT and robot
tasks) or into a series of ADT and robot tasks (which is simply all
the ADT and robot tasks for the passes taken in order). This schema
does not formalize the decomposition into passes. It may turn out to
be desirable to do this, but it does not now seem necessary.

*)
```

```
ENTITY chamfer_feature SUBTYPE OF (adacs_e_move);
      chamfer_to_make:        chamfer_face;
      spindle_speed:                INTEGER;
      chamfer_feature_plan_id :     OPTIONAL plan_id;
WHERE
        0 <= spindle_speed;
        spindle_speed <= 90000;
END_ENTITY;

(* chamfer_list*****************************************************************

The list of chamfer_faces given here is expected to contain entity
references from the design. However, the design is expected to be kept
separate from the chamfer_list (in a separate file, for example).  A
convention for identifying physical files (or for partitioning a
database) will be required so that the two files (or database parts)
can work together.

The lower bound for number of faces in a chamfer_list might be raised
to 1, but it seems convenient to be able to write an empty list -
might be useful in dry runs.

*)

ENTITY chamfer_list;
      part_design:            design_id;
      chamfers:         LIST [0:?] OF chamfer_face;
END_ENTITY;

(* chamfer_part*****************************************************************

part_id - identifies a specific workpiece to be chamfered. There may
be many workpieces made according to the same design. This attribute
must have a value in a command, but does not need one in a process
plan.  The description of the workpiece that may be retrieved from
knowing its part_id should include identification of its design and
its location. The design of the workpiece should be the same as the
design designated by the part_design attribute.

part_design - identifies the design for the part being chamfered. It
is expected that STEP boundary representation designs will be used and
may be identified by a design_id.

tool_type - identifies a specific type of tool (like a catalog number).
It is expected that the tool will not be changed during the chamfering
of the part. If more than one tool is to be used, separate chamfer_part
tasks will be required. Since this attribute is given here, it is not
needed in the chamfer_feature entity.

tool - identifies a specific physical tool (like an inventory number).
This is optional and is expected to have a value in a command but not
normally in a process plan.

chamfers - A description of the chamfers to make.
```

```
chamfer_part_plan_id - If a process plan has already been made giving a
series of tasks for the E-move level that will chamfer the part, this
identifies it. If the controller executing a chamfer_part command is
expected to determine tasks for subordinates in real time, this
attribute has no value.

Some information might be expected in this task but is not included:
1. The location of the workpiece - this may be retrieved by knowing
   the part_id.
2. The fixturing of the part - If a chamfer_part_plan has been prepared,
   fixturing specifications will be included as part of the plan. If a
   plan has not been prepared, it is expected that a decision on how to
   fixture the part will not have been made.

*)

ENTITY chamfer_part SUBTYPE OF (adacs_task);
      part_id:          OPTIONAL workpiece_id;
      part_design:            design_id;
      tool_type:        tool_type_id;
      tool:             OPTIONAL tool_id;
      chamfers:         chamfer_instruction;
      chamfer_part_plan_id:   OPTIONAL plan_id;
END_ENTITY;

(* design_id  ************************************************************

This identifies the name of the design and its version. It is expected
that this information will be sufficient to select the design from a
design database.

It might be nice to replace this with an Entity that serves the same
purpose from STEP Application Protocol 203, Configuration Controlled
Design.

*)

ENTITY design_id;
      name:       STRING;
      version:    STRING;
END_ENTITY;


(* fixture_part ***********************************************************

This command is given to the E-move level by the Task level. It is to
be executed by a human.

It is assumed that given the part and the fixture, there are no
further degrees of freedom, and the the human will know how to do the
job. If there are more degrees of freedom, such as position within the
fixture, this entity will have to be changed. An attribute might be
added whose value is a string giving fixturing instructions.
Alternatively, there might be an attribute that identified a file of
```

```
fixturing instructions.

get_from - This string should describe in English where to get the part
from.

*)

ENTITY fixture_part SUBTYPE OF (adacs_e_move);
        part_id:  workpiece_id;
      fixture:    fixture_id;
      get_from:   STRING;
END_ENTITY;


(* t3_control_off ************************************************************

*)

ENTITY t3_control_off SUBTYPE OF (t3_servo);
END_ENTITY;

(* t3_control_on  ************************************************************

*)

ENTITY t3_control_on SUBTYPE OF (t3_servo);
END_ENTITY;

(* t3_move *******************************************************************

All T3 moves require a trajectory. In a t3_move, the robot simply
follows the trajectory from the beginning to the end. It is expected
that the robot will already be in the first pose of the trajectory
when this command is given.

It is intended that t3_move be instantiable, even though it has a
subtype (t3_move_until), so the format of the supertype statement
differs from that of other supertype statements.

*)

ENTITY t3_move
  SUPERTYPE OF (t3_move_until)
  SUBTYPE OF (t3_primitive);
        robot_path:      trajectory;
END_ENTITY;

(* t3_move_to  ***************************************************************

This moves the robot from its current position to the given pose. It
does not specify velocity or acceleration when the pose is reached or
during the move.

*)
```

```
ENTITY t3_move_to SUBTYPE OF (t3_servo);
        robot_position:      pose;
END_ENTITY;


(* t3_move_until  ***********************************************************

In this command the robot moves along the given trajectory until either
it reaches the end or until the ADT force vector goes outside a
rectangular envelope.

*)

ENTITY t3_move_until SUBTYPE OF (t3_Move);
      low_normal:  INTEGER;
      high_normal:       INTEGER;
        low_tangential:  INTEGER;
        high_tangential: INTEGER;
WHERE
      0 <= low_normal;
      low_normal <= 3000;
      0 <= high_normal;
      high_normal <= 3000;
        0 <= low_tangential;
      low_tangential <= 3000;
      0 <= high_tangential;
      high_tangential <= 3000;
END_ENTITY;

(* t3_power_off *************************************************************

*)

ENTITY t3_power_off SUBTYPE OF (t3_servo);
END_ENTITY;

(* t3_power_on  *************************************************************

*)

ENTITY t3_power_on SUBTYPE OF (t3_servo);
END_ENTITY;

(* t3_primitive *************************************************************

This is subtype of adacs_primitive to be directed to the T3 robot. It serves
only as a supertype for other Primitive level robot tasks.

*)

ENTITY t3_primitive
  SUPERTYPE OF (ONEOF
                   (t3_move,
                    t3_shutdown,
                    t3_startup))
```

```
  SUBTYPE OF (adacs_primitive);
END_ENTITY;

(* t3_servo ***************************************************************

This is subtype of adacs_servo to be directed to the T3 robot. It serves
only as a supertype for other Servo level robot tasks.

*)

ENTITY t3_servo
  SUPERTYPE OF (ONEOF
                   (t3_control_off,
                    t3_control_on,
                    t3_move_to,
                    t3_power_off,
                    t3_power_on))
  SUBTYPE OF (adacs_servo);
END_ENTITY;

(* t3_shutdown ************************************************************

This shuts down the Primitive level T3 robot controller. After this
command has been received, the only command the controller will
execute is a startup. Normally this command will be followed by
powering down.

*)

ENTITY t3_shutdown SUBTYPE OF (t3_primitive);
END_ENTITY;

(* t3_startup *************************************************************

This starts up the Primitive level T3 robot controller after it has
been powered up and reached a state where it can read commands. This
must be the first command given to the controller after it reaches
that state.  Any other command will be ignored.

*)

ENTITY t3_startup SUBTYPE OF (t3_primitive);
END_ENTITY;

(* unfixture_part *********************************************************

This command is given to the E-move level by the Task level. It is to
be executed by a human.  When this command is given, the part is
removed from the fixture and put in the location described by the put_at
string.

put_at - This string describes in English where to put the part after it
is removed from the fixture.

*)
```

```
ENTITY unfixture_part SUBTYPE OF (adacs_e_move);
        part_id:          workpiece_id;
      put_at:             STRING;
END_ENTITY;


(* END ADACS ENTITY DEFINITIONS *)
(*****************************************************************************)
(*****************************************************************************)

END_SCHEMA;

(* END ADACS_ACTIONS SCHEMA *)


(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)
(* START ADACS_GEOMETRY SCHEMA *)

(*

This schema contains three entities for robot positioning and a bunch
of entities for describing chamfer_faces.

The rest of the text here, before EXPRESS statements start, explains
the meaning of a chamfer_face, its subtypes, and the entities needed
to describe a chamfer_face. In this text, "face", "surface", "edge",
"edge_logical_structure", "curve", and "curve_logical_structure" will
have the meanings given by STEP geometry and topology.  A surface
produced by chamfering will be called a "chamfer_surface" in the text,
although chamfer_surface is not defined in the EXPRESS statements.

An informal description of the entities used here from STEP geometry
and topology follows. Their EXPRESS description is given below.

curve - A curve is the path of a point moving in space. Straight lines
and circles are examples of curves. Every curve has a built-in
direction of travel along the curve which can be determined from the
data describing the curve.

curve_logical_structure - A curve_logical_structure is a curve with a
direction flag which can be true or false. If the flag is true, you
are moving in the direction of the curve. If the flag is false you are
moving in the opposite direction.

edge - An edge is a piece of a curve bounded by two vertexes (points).
It is defined with a curve_logical_structure to indicate whether
you go in the same direction as the curve when moving along the curve
from the start vertex to the end vertex. The start and end may be in
the same place, as would be the case if you went all the way around a
circle.

edge_logical_structure - An edge_logical_structure is an edge with a
```

direction flag which can be true or false. If the flag is true, you are moving along the edge from the start vertex to the end vertex. If the flag is false you are moving from the end vertex to the start vertex.

face - A face is a piece of a surface bounded by a loop of edge_logical_structures.

surface - A surface is a two-dimensional sheet in 3D space. Planes and the outer boundaries of spheres and cylinders are examples of surfaces.

Intuitively, a chamfer_face is the flattish face (or faces) produced by scraping material off an edge (or edges) using a straight-edged tool. We will use a logical procedural description of a chamfer-face. Such a description is useful in determining what operations are required to make the chamfer_face, but the logical procedures should not be confused with procedures for using a chamfer tool.

A boundary representation description of a chamfer_face is apt to require a great deal of complex data and is not much use in determining how to make the chamfer_face, so we usually will want to avoid using such a description. If a boundary representation is desirable (when a closeup of a chamfer_face is to be drawn, for example), it will be logically possible to produce it, although it may be computationally difficult.

The data defining a chamfer_face consists of a list of edge_logical_structures (the list may contain only one element), plus the depth of the chamfer and specification of a normal vector at each point of the edge(s). In addition, if the chamfer_face is open (i.e. it does not form a loop), specifications for ramping from zero depth to the given depth at the beginning of the chamfer_face and back out again at the end may optionally be given.

It is expected that the edge_logical_structures will be present in the boundary representation of the unchamfered part. The elements of the list of edge_logical_structures must represent directed edges that meet with the end vertex of element n being the start vertex of element n+1. If the chamfer_face is closed, the start vertex of the first element must also be the end vertex of the last element.

All chamfer_surfaces will be ruled surfaces (planes, cones, and cylinders, for example) where the ruling lies in a plane perpendicular to the edge being chamfered. A ruled surface is one that can be swept out by a straight line moving along a path. The nominal surfaces made by a conical or cylindrical chamfer tool used in typical chamfering operations are usually ruled surfaces of the sort just described. (Actual surfaces are generally more rounded than nominal surfaces if the chamfering is done by hand.)

It is expected that each chamfer will have constant depth. The depth is measured from the path in the direction of the chamfer_surface normal.

Open and closed subtypes of chamfer_face are given in this schema.

The standard fixed rule for the normal to a chamfer_surface will be: at each point on an edge being chamfered, the normal to the chamfer surface will lie in a plane perpendicular to the edge and will make equal angles with the normals to the surfaces on either side of the path at that point. This is the most usual way of making a chamfer. This rule will need some elaboration regarding the treatment of sharp corners of a path and treatment of joints of the path where the direction of the normal to the chamfer_surface would jump discontinuously from one value to another.

Only edges between surfaces which meet convexly can be chamfered.

It would be possible to have other methods of specifying chamfers such as by allowing variable depth or different rules regarding the normal to the chamfer_surface, but these do not seem needed. The simplest elaboration would be to have a parameter to specify the ratio of (1) the angle of the chamfer_surface normal with the normal to the surface on one side of the edge being chamfered to (2) the angle of the chamfer_surface normal with the normal to the surface on the other side. The standard rule given above has this ratio always equal to one.

*)

```
SCHEMA adacs_geometry;

(*****************************************************************************)
(*****************************************************************************)

(* START ADACS_GEOMETRY INTERFACE CLAUSES *)

USE FROM geometry_schema (circle, direction, line);

USE FROM topology_schema
      (curve_logical_structure,
       edge_logical_structure);

(* END ADACS_GEOMETRY INTERFACE CLAUSES *)
(*****************************************************************************)
(*****************************************************************************)
(* START ADACS_GEOMETRY ENTITY DEFINITIONS *)


(* chamfer_face ************************************************************
```

See the discussion at the beginning of this schema.

normal_spec - It is not expected that normal_spec will need to be used in the ADACS project. It is included here for generality. The length of the normal_spec list, if used, must be the same as the length of the chamfer_path list (could add a "where" rule that says this), and the nth elements of the two lists go together. The string which is the nth element of the normal_spec list would be a real-valued

function of one real variable with domain from 0 to 1. The value of the function would give the deviation in radians of the normal to the path from its standard direction, as described above. The normal would always lie in a plane perpendicular to the edge at a point on the edge. The angle would be positive in the counterclockwise direction, when facing in the positive direction of the edge_logical_structure and looking at that plane. The value of the variable in the function would represent the fraction of the distance from the start vertex to the end vertex of the point on the nth edge_logical_structure for which the normal is given by the equation.

If it is desired to make the depth of the chamfer variable, a "depth_spec" attribute could be added with the same format as the "normal_spec" attribute. The value of the function would be the depth of the chamfer.

It might be nice to use a unitted number rather than a real number for the value of the depth attribute, since otherwise there will need to be a convention about the units of depth. Of course a convention is needed for the units of the boundary representation already.

The procedure for constructing a chamfer_face is as follows. For each point Q on an edge being chamfered, let N be the chamfer_surface normal at Q (determined either by the standard rule or by the value of the normal_spec attribute). Let P be the plane of N and the tangent to the edge at Q. At a distance D (the depth of the chamfer) from Q in the direction opposite N (into the material), construct a line perpendicular to P. Where the line intersects the surfaces on either side of the edge, cut the line off. The set of points on all such lines comprises the chamfer_face.

The hierarchy of chamfer_faces is as follows:

```
chamfer_face
      chamfer_face_closed
            chamfer_face_closed_advanced
            chamfer_face_closed_elementary
                  chamfer_face_closed_elementary_circle
                  chamfer_face_closed_elementary_other
      chamfer_face_open
            chamfer_face_open_advanced
            chamfer_face_open_elementary
                  chamfer_face_open_elementary_arc
                        chamfer_face_open_elementary_line
                  chamfer_face_open_elementary_other

*)

ENTITY chamfer_face
  SUPERTYPE OF (ONEOF
                  (chamfer_face_closed,
                   chamfer_face_open));
      depth:              REAL;
      chamfer_path:  LIST [1 : ?] OF edge_logical_structure;
      normal_spec:   OPTIONAL LIST [1 : ?] OF STRING;
```

```
END_ENTITY;

(* chamfer_face_closed*****************************************************

This is a chamfer_face produced by following a list of
edge_logical_structures that form a closed loop.

It is a separate subtype since it is expected that the technique for
chamfering a closed loop will differ from the technique for chamfering
an open path.

*)

ENTITY chamfer_face_closed
  SUPERTYPE OF (ONEOF
                  (chamfer_face_closed_elementary,
                 chamfer_face_closed_advanced))
  SUBTYPE OF (chamfer_face);
END_ENTITY;

(* chamfer_face_closed_advanced ********************************************

This is a chamfer_face_closed in which one or more of the elements of
the chamfer_path does not have a line or circle as its underlying curve
type.

*)

ENTITY chamfer_face_closed_advanced SUBTYPE OF (chamfer_face_closed);
END_ENTITY;

(* chamfer_face_closed_elementary ******************************************

This is a chamfer_face_closed in which all of the elements of the
chamfer_path have a line or circle as their underlying curve type.

*)

ENTITY chamfer_face_closed_elementary
  SUPERTYPE OF (ONEOF
                  (chamfer_face_closed_elementary_circle,
                   chamfer_face_closed_elementary_other))
  SUBTYPE OF (chamfer_face_closed);
END_ENTITY;

(* chamfer_face_closed_elementary_circle ***********************************

This is a chamfer_face produced by following a circular closed path.

The chamfer_path must have only one element. The curve underlying that
element must be a circle, and the start and end vertexes of the edge
underlying that element must be the same (implying that the edge
consists of the entire circle).

The third part of the "where" clause is messy because there are two
```

```
ways of describing an edge - the edge_curve may be a curve or a
curve_logical_structure.

*)

ENTITY chamfer_face_closed_elementary_circle
  SUBTYPE OF (chamfer_face_closed_elementary);
WHERE
(* length of chamfer_path is 1 *)
      SIZEOF (chamfer_path) = 1;

(* vertexes at ends are the same *)
      chamfer_path[1].edge_element.edge_start
         = chamfer_path[1].edge_element.edge_end;

(* the curve is a circle *)
      ((curve_logical_structure
            IN TYPEOF (chamfer_path[1].edge_element.edge_curve)) AND
        (circle
            IN TYPEOF
(chamfer_path[1].edge_element.edge_curve.curve_element)))
      OR
       (circle IN TYPEOF (chamfer_path[1].edge_element.edge_curve));
END_ENTITY;


(* chamfer_face_closed_elementary_other ***********************************

This is a chamfer_face_closed_elementary in which the chamfer_path is
not a circle.

*)

ENTITY chamfer_face_closed_elementary_other
   SUBTYPE OF (chamfer_face_closed_elementary);
END_ENTITY;

(* chamfer_face_open ******************************************************

This is a chamfer_face produced by following an open path. It has
optional end conditions for ramping in and out. If an end condition is
not specified at an end, the chamfer is at full depth near that end.

If one of chamfer_start_out and chamfer_start_in is present, both must
be present.

If one of chamfer_end_in and chamfer_end_out is present, both must
be present.

The following must hold:

0.0 <= chamfer_start_out < chamfer_start_in <= 1.0   if present
0.0 <= chamfer_end_in    < chamfer_end_out  <= 1.0   if present

There is probably a way to say that using "where" rules.
```

```
The idea of chamfer_start_out and chamfer_start_in is that they
represent locations along the first edge_logical_structure of the
chamfer_path at which ramping into the edge begins and ends. The value
of the attribute represents the fraction of the distance along the
edge.  The actual shape of the ramp needs to be specified by a
convention that gives shapes that can actually be made by a chamfer
tool. It the curve of the edge is a line, the ramp surface will be
cylindrical, and if the curve is a circle, the ramp surface will be
conical (possibly degenerating to a cylinder or plane). In either
case, the surface will pass through the chamfer_start_out point and
the line on the chamfer_face generated by the chamfer_start_in point.
If the curve is some other curve, the shape of the ramp will be
unspecified.

chamfer_end_out and chamfer_end_in are handled analogously to
chamfer_start_out and chamfer_start_in, except that they apply to
the last edge_logical_structure of the chamfer_path.

It is permissible for there to be only element in the chamfer_path.
In this case it is both the first element and the last element.  If
all four optional attributes are present in this case, we must have
chamfer_start_in <= chamfer_end_in, as well as the inequalities above.

If a normal_spec is used, the value of the first function in the list
must be 0 if a chamfer_start is used, and the value of the last function
in the list must be 0 if a chamfer_end is used. A zero value for the
function simply implies that the standard rule for the normal is
used.

Note that if a chamfer_start or chamfer_end is used, the in and out
points must be far enough apart to accommodate the ramp surface. This
could be a problem in chamfering detailed parts.

*)

ENTITY chamfer_face_open
  SUPERTYPE OF (ONEOF
                  (chamfer_face_open_advanced,
                   chamfer_face_open_elementary))
  SUBTYPE OF (chamfer_face);
      chamfer_start_out:  OPTIONAL REAL;
        chamfer_start_in:   OPTIONAL REAL;
        chamfer_end_in:     OPTIONAL REAL;
        chamfer_end_out:    OPTIONAL REAL;
END_ENTITY;

(* chamfer_face_open_advanced ********************************************

This is a chamfer_face_open in which one or more of the elements of
the chamfer_path does not have a line or circle as its underlying curve
type.

*)
```

```
ENTITY chamfer_face_open_advanced SUBTYPE OF (chamfer_face_open);
END_ENTITY;

(* chamfer_face_open_elementary ********************************************

This is a chamfer_face_open in which all of the elements of the
chamfer_path have a line or circle as their underlying curve type.

*)

ENTITY chamfer_face_open_elementary
  SUPERTYPE OF (ONEOF
                  (chamfer_face_open_elementary_arc,
                   chamfer_face_open_elementary_line,
                   chamfer_face_open_elementary_other))
  SUBTYPE OF (chamfer_face_open);
END_ENTITY;

(* chamfer_face_open_elementary_arc ****************************************

This is a chamfer_face produced by following an open path along a
circular arc.

The chamfer_path must have only one element. The curve underlying that
element must be a circle.

The second part of the "where" clause is messy because there are two
ways of describing an edge - the edge_curve may be a curve or a
curve_logical_structure.

It is legal for the start and end of the arc to be in the same place,
but usually the arc will be less than a full circle.

*)

ENTITY chamfer_face_open_elementary_arc
  SUBTYPE OF (chamfer_face_open_elementary);
WHERE
(* length of chamfer_path is 1 *)
        SIZEOF (chamfer_path) = 1;

(* the curve is a circle *)
        ((curve_logical_structure
            IN TYPEOF (chamfer_path[1].edge_element.edge_curve)) AND
          (circle
            IN TYPEOF
(chamfer_path[1].edge_element.edge_curve.curve_element)))
        OR
          (circle IN TYPEOF (chamfer_path[1].edge_element.edge_curve));
END_ENTITY;

(* chamfer_face_open_elementary_line ***************************************

This is a chamfer_face produced by following a line segment.
```

```
The second part of the "where" clause is messy because there are two
ways of describing an edge - the edge_curve may be a curve or a
curve_logical_structure.

*)

ENTITY chamfer_face_open_elementary_line
  SUBTYPE OF (chamfer_face_open_elementary);
WHERE
(* length of chamfer_path is 1 *)
        SIZEOF (chamfer_path) = 1;

(* the curve is a line *)
        ((curve_logical_structure
            IN TYPEOF (chamfer_path[1].edge_element.edge_curve)) AND
          (line
            IN TYPEOF
(chamfer_path[1].edge_element.edge_curve.curve_element)))
        OR
          (line IN TYPEOF (chamfer_path[1].edge_element.edge_curve));
END_ENTITY;

(* chamfer_face_open_elementary_other ************************************

This is a chamfer_face_open_elementary in which the chamfer_path is
not a single arc or line.

*)

ENTITY chamfer_face_open_elementary_other
   SUBTYPE OF (chamfer_face_open_elementary);
END_ENTITY;

(* pose ******************************************************************

A pose gives the position and orientation of a robot end-effector.
The "direction" type of value is from the STEP geometry schema.
The definition of this entity may change.

*)

ENTITY pose;
      translation:      direction;
      rotation:   quaternion;
END_ENTITY;

(* quaternion ************************************************************

A quaternion specifies an axis of rotation and an amount of rotation.

The x, y, and z components describe a unit vector in the direction of
the axis of rotation.

s is the sine of half the angle of rotation.
```

```
The "where" clause is worded with exact equality. An implementation
would have to allow a range.

*)

ENTITY quaternion;
      s:    REAL;
      x:    REAL;
      y:    REAL;
      z:    REAL;
WHERE
      -1.0 < s;
      s <= 1.0;
      ((x**2) + (y**2) + (z**2)) = 1.0;
END_ENTITY;

(* trajectory ********************************************************

A trajectory is a series of robot poses.

*)

ENTITY trajectory;
      elements:   LIST [1 : ?] OF pose;
END_ENTITY;

(* END ADACS_GEOMETRY ENTITY DEFINITIONS *)
(*****************************************************************************)
(*****************************************************************************)

END_SCHEMA;

(* END ADACS_GEOMETRY SCHEMA *)


(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)
(* START GEOMETRY SCHEMA *)

(*

This includes geometry entities needed by the adacs_geometry schema
and/or the STEP topology schema. All "where" and "derive" clauses and
unused subtypes have been deleted.  length_measure has been changed to
REAL.

Definitions of more advanced types of curves and surfaces will have to
be added to this to handle parts such as turbine blades and rotor hubs.

*)


SCHEMA geometry_schema;
```

```
(*****************************************************************************)
(*****************************************************************************)
(* START GEOMETRY ENTITY DEFINITIONS *)

ENTITY geometry
  SUPERTYPE OF (ONEOF(point, vector, axis_placement, curve, surface));
END_ENTITY;

ENTITY point
  SUPERTYPE OF (cartesian_point)
  SUBTYPE OF (geometry);
END_ENTITY;

ENTITY cartesian_point
  SUBTYPE OF (point);
  x_coordinate : REAL;
  y_coordinate : REAL;
  z_coordinate : OPTIONAL REAL;
END_ENTITY;

ENTITY vector
  SUPERTYPE OF (direction)
  SUBTYPE OF (geometry);
END_ENTITY;

ENTITY direction
  SUBTYPE OF (vector);
  x  :  REAL;
  y  :  REAL;
  z  :  OPTIONAL REAL;
END_ENTITY;

ENTITY axis_placement
  SUPERTYPE OF (axis2_placement)
  SUBTYPE OF (geometry);
END_ENTITY;

ENTITY axis2_placement
  SUBTYPE OF (axis_placement);
  location      : cartesian_point;
  axis          : OPTIONAL direction;
  ref_direction : OPTIONAL direction;
END_ENTITY;

ENTITY curve
  SUPERTYPE OF (ONEOF(line, conic))
  SUBTYPE OF (geometry);
END_ENTITY;

ENTITY line
  SUBTYPE OF (curve);
  pnt     : cartesian_point;
  dir     : direction;
END_ENTITY;
```

```
ENTITY conic
  SUPERTYPE OF (ONEOF(circle, ellipse))
  SUBTYPE OF (curve);
END_ENTITY;

ENTITY circle
  SUBTYPE OF (conic);
  radius   : REAL;
  position : axis2_placement;
END_ENTITY;

ENTITY ellipse
  SUBTYPE OF (conic);
  semi_axis_1 : REAL;
  semi_axis_2 : REAL;
  position    : axis2_placement;
END_ENTITY;

ENTITY surface
  SUPERTYPE OF (elementary_surface)
  SUBTYPE OF (geometry);
END_ENTITY;

ENTITY elementary_surface
  SUPERTYPE OF (ONEOF(plane, cylindrical_surface, conical_surface,
                      spherical_surface, toroidal_surface))
  SUBTYPE OF (surface);
END_ENTITY;

ENTITY plane
  SUBTYPE OF (elementary_surface);
  position : axis2_placement;
END_ENTITY;

ENTITY cylindrical_surface
  SUBTYPE OF (elementary_surface);
  radius     : REAL;
  position   : axis2_placement;
END_ENTITY;

ENTITY conical_surface
  SUBTYPE OF (elementary_surface);
  semi_angle : REAL;
  radius     : REAL;
  position   : axis2_placement;
END_ENTITY;

ENTITY spherical_surface
  SUBTYPE OF (elementary_surface);
  radius   : REAL;
  position : axis2_placement;
END_ENTITY;

ENTITY toroidal_surface
  SUBTYPE OF (elementary_surface);
```

```
  major_radius : REAL;
  minor_radius : REAL;
  position     : axis2_placement;
END_ENTITY;

(* END GEOMETRY ENTITY DEFINITIONS *)
(*****************************************************************************)
(*****************************************************************************)

END_SCHEMA;

(* END GEOMETRY SCHEMA *)


(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)
(* START TOPOLOGY SCHEMA *)

SCHEMA topology_schema;

(*****************************************************************************)
(*****************************************************************************)
(* START TOPOLOGY INTERFACE CLAUSES *)

REFERENCE FROM geometry_schema;

(* END TOPOLOGY INTERFACE CLAUSES *)
(*****************************************************************************)
(*****************************************************************************)
(* START TOPOLOGY TYPE DEFINITIONS *)

TYPE curve_or_logical = SELECT (curve, curve_logical_structure);
END_TYPE;

TYPE edge_or_logical = SELECT (edge, edge_logical_structure);
END_TYPE;

TYPE loop_or_logical = SELECT (loop, loop_logical_structure);
END_TYPE;

TYPE surface_or_logical = SELECT (surface, surface_logical_structure);
END_TYPE;

TYPE face_or_logical = SELECT (face, face_logical_structure);
END_TYPE;

(* END TOPOLOGY TYPE DEFINITIONS *)
(*****************************************************************************)
(*****************************************************************************)
(* START TOPOLOGY ENTITY DEFINITIONS *)

ENTITY topology
  SUPERTYPE OF (ONEOF (vertex, edge, loop, face, shell));
END_ENTITY;
```

```
ENTITY vertex
  SUBTYPE OF (topology);
  vertex_point : OPTIONAL point;
END_ENTITY;

ENTITY edge
  SUBTYPE OF (topology);
  edge_start   : vertex;
  edge_end     : vertex;
  edge_curve   : OPTIONAL curve_logical_structure;
END_ENTITY;

ENTITY curve_logical_structure;
  curve_element : curve;
  flag          : BOOLEAN;
END_ENTITY;

ENTITY edge_logical_structure;
  edge_element : edge;
  flag : BOOLEAN;
END_ENTITY;

ENTITY loop
  SUPERTYPE OF (ONEOF (vertex_loop, edge_loop))
  SUBTYPE OF (topology);
END_ENTITY;

ENTITY vertex_loop
  SUBTYPE OF (loop);
  loop_vertex  : vertex;
END_ENTITY;

ENTITY edge_loop
  SUBTYPE OF (loop);
  loop_edges : LIST [1 : ?] OF edge_or_logical;
END_ENTITY;

ENTITY face
  SUBTYPE OF (topology);
  outer_bound  : OPTIONAL loop_or_logical;
  bounds       : SET [1 : ?] OF loop_or_logical;
  face_surface : OPTIONAL surface_or_logical;
END_ENTITY;

ENTITY loop_logical_structure;
  loop_element : loop;
  flag         : BOOLEAN;
END_ENTITY;

ENTITY surface_logical_structure;
  surface_element : surface;
  flag            : BOOLEAN;
END_ENTITY;
```

```
ENTITY shell
  SUPERTYPE OF (closed_shell)
  SUBTYPE OF (topology);
END_ENTITY;

ENTITY face_logical_structure;
  face_element : face;
  flag         : BOOLEAN;
END_ENTITY;

ENTITY closed_shell
  SUBTYPE OF (shell);
  shell_boundary  : SET [1 : ?] OF face_or_logical;
END_ENTITY;

ENTITY shell_logical_structure;
  shell_element : shell;
  flag          : BOOLEAN;
END_ENTITY;

(* END TOPOLOGY ENTITY DEFINITIONS *)
(*****************************************************************************)
(*****************************************************************************)

END_SCHEMA;

(* END TOPOLOGY SCHEMA *)
(*****************************************************************************)
(*****************************************************************************)
(*****************************************************************************)
```

NIST-114A
(REV. 3-90)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

# BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NISTIR 4888

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
JULY 1992

4. TITLE AND SUBTITLE

NIST Support to the Next Generation Controller Program: 1991 Final Technical Report

5. AUTHOR(S)

James Albus, Richard Quintero, Frederick Proctor, Thomas Kramer, John Michaloski, Nicholas Dagalakis, Nicholas Tarnoff

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

10. SUPPLEMENTARY NOTES

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

This report covers work performed by the National Institue of Standards and Technology (NIST) Robot Systems Division between October 1, 1990 and December 31, 1991 in continuation of work previously undertaken by NIST during FY90 for the U.S. Air Force Next Generation Controller Program (NGC).

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

Next Generation Controller (NGC); Open Architecture; Neutral Manufacturing Language (NML); Real-time Control System (RCS)

13. AVAILABILITY

[X] UNLIMITED

[ ] FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).

[ ] ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.

[ ] ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.

14. NUMBER OF PRINTED PAGES
196

15. PRICE
A09

ELECTRONIC FORM