NISTIR 6910

4D/RCS: A Reference Model Architecture For Unmanned Vehicle Systems Version 2.0

[a]James Albus, Hui-Min Huang, Elena Messina, Karl Murphy, Maris Juberts, Alberto Lacaze, Stephen Balakirsky, Michael Shneier, Tsai Hong, Harry Scott, Frederick Proctor, William Shackleford, John Michaloski, Albert Wavering, Thomas Kramer, Nicholas Dagalakis, William Rippey, Keith Stouffer, Steven Legowik, John Evans, Roger Bostelman, Richard Norcross, Adam Jacoff, Sandor Szabo, Joseph Falco, Robert Bunch, James Gilsinn, Tommy Chang, Tsung-Ming Tsai

[b]Alexander Meystel
[c]Anthony Barbera, M.L. Fitzgerald
[d]Mark Del Giorno
[e]Robert Finkelstein

[a]U. S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Intelligent Systems Division
Gaithersburg, MD 20899-8230

[b]Drexel University and NIST
[c]Advanced Technology & Research Corp.
[d]General Dynamics Robotic Systems, Inc.
[e]Robotic Technology, Inc.

NIST
National Institute of Standards and Technology
Technology Administration
U.S. Department of Commerce

4D/RCS: A Reference Model Architecture For Unmanned Vehicle Systems Version 2.0

NISTIR 6910

[a]James Albus, Hui-Min Huang, Elena Messina, Karl Murphy, Maris Juberts, Alberto Lacaze, Stephen Balakirsky, Michael Shneier, Tsai Hong, Harry Scott, Frederick Proctor, William Shackleford, John Michaloski, Albert Wavering, Thomas Kramer, Nicholas Dagalakis, William Rippey, Keith Stouffer, Steven Legowik, John Evans, Roger Bostelman, Richard Norcross, Adam Jacoff, Sandor Szabo, Joseph Falco, Robert Bunch, James Gilsinn, Tommy Chang, Tsung-Ming Tsai

[b]Alexander Meystel
[c]Anthony Barbera, M.L. Fitzgerald
[d]Mark Del Giorno
[e]Robert Finkelstein

[a]U. S. DEPARTMENT OF COMMERCE
Technology Administration
National Institute of Standards and Technology
Intelligent Systems Division
Gaithersburg, MD 20899-8230

[b]Drexel University and NIST
[c]Advanced Technology & Research Corp.
[d]General Dynamics Robotic Systems, Inc.
[e]Robotic Technology, Inc.

August 2002

U.S. DEPARTMENT OF COMMERCE
Donald L. Evans, Secretary

TECHNOLOGY ADMINISTRATION
Phillip J. Bond, Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS
AND TECHNOLOGY
Arden L. Bement, Jr., Director

4D/RCS:
A Reference Model Architecture For Unmanned Vehicle Systems
Version 2.0

**James Albus, Hui-Min Huang, Elena Messina, Karl Murphy,
Maris Juberts, Alberto Lacaze, Stephen Balakirsky, Michael Shneier,
Tsai Hong, Harry Scott, Frederick Proctor, William Shackleford,
John Michaloski, Albert Wavering, Thomas Kramer, Nicholas Dagalakis,
William Rippey, Keith Stouffer, Steven Legowik, John Evans,
Roger Bostelman, Richard Norcross, Adam Jacoff, Sandor Szabo,
Joseph Falco, Robert Bunch, James Gilsinn, Tommy Chang, Tsung-Ming Tsai**
Intelligent Systems Division
National Institute of Standards and Technology

Alexander Meystel
Drexel University and NIST

Anthony Barbera
M.L. Fitzgerald
Advanced Technology & Research Corp.

Mark Del Giorno
General Dynamics Robotic Systems, Inc.

Robert Finkelstein
Robotic Technology, Inc.

Prepared for:
The Army Research Laboratory
Demo III Program
Charles Shoemaker, Program Manager

August 2002
National Institute of Standards and Technology
Gaithersburg, Maryland 20899

ABSTRACT

The 4D/RCS architecture provides a reference model for military unmanned vehicles on how their software components should be identified and organized. It defines ways of interacting to ensure that missions, especially those involving unknown or hostile environments, can be analyzed, decomposed, distributed, planned, and executed intelligently, effectively, efficiently and in coordination. To achieve this, the 4D/RCS reference model provides well defined and highly coordinated sensory processing, world modeling, knowledge management, cost/benefit analysis, behavior generation, and messaging functions, as well as the associated interfaces. The 4D/RCS architecture is based on scientific principles and is consistent with military hierarchical command doctrine.

The 4D/RCS reference model architecture is naturally adaptable to the DoD/Army standards in a combined domain of vehicle systems, combat support, and software engineering. 4D/RCS provides an architectural framework to facilitate component and interface standards development, including command and control, sensors, communication, mapping, operating environments, safety, security, software engineering, user interface, data interchange, and graphics. As such, the 4D/RCS reference model architecture forms a framework for software engineering standards and guidelines.

TABLE OF CONTENTS

1.0 INTRODUCTION 9
Standards Orientation 10
Science and Technology Orientation 11
Domains 11
1.1 SCOPE 12
1.1.1 A Conceptual Framework 12
1.1.2 A Reference Model Architecture 13
1.1.3. Engineering Guidelines 13
Software Reuse 14
Interoperability and open systems 15
2.0 A CONCEPTUAL FRAMEWORK 16
3.0 4D/RCS REFERENCE MODEL ARCHITECTURE 19
3.1 THE 4D/RCS HIERARCHY 19
3.2 SENSORY PROCESSING 23
3.3 WORLD MODELING 24
3.4 VALUE JUDGMENT 26
3.5 BEHAVIOR GENERATION 26
3.6 THE RCS_NODE 27
3.7 AN ORGANIZATION OF RCS_NODES 28
3.8 HIERARCHICAL LEVELS 31
3.9 FOCUS OF ATTENTION 33
3.10 A NOTIONAL 4D/RCS CONCEPT FOR FCS 34
3.11 4D/RCS FOR DEMO III 42
3.12 THE INTER-NODE INTERACTIONS WITHIN A HIERARCHY 45
3.13 COMMAND VOCABULARIES 46
3.14 COMMAND AND PLAN STRUCTURE 49
3.15 REPLANNING 55
3.16 TWO KINDS OF PLANS 56
4.0 ENGINEERING GUIDELINES 58
4.1 BEHAVIOR GENERATION 58
4.2 WORLD MODELING 82
4.3 VALUE JUDGMENT 84
4.4 KNOWLEDGE DATABASE 85
4.4.1 Signals 86
4.4.2 Entities 86
4.4.3 Classes of Entities 92
4.4.4 Images 97
4.4.5 Images and Frames 100
4.4.6 Image Field of View 101
4.4.7 Maps 105
4.4.8 Events 107
4.4.9 Event Grouping 111
4.4.10 Timing 112

4.4.11 Temporal Persistence of Representation 114
4.4.12 Knowledge of Rules of Mathematics and Logic 118
4.4.13 Knowledge of Rules of Physics 119
4.4.14 Knowledge of Value 120
4.5 SENSORY PROCESSING 120
4.5.1 Sensor Processing (SP) Functionality 121
4.5.2 A Typical Processing Node 128
4.5.3 Hypothesize and Test 130
4.5.4 Recursive Estimation 132
4.5.5 Pyramids of Scale 139
4.5.6 Levels of Sensory Processing 140
5.0 GENERIC SHELL 156
6.0 SUMMARY AND CONCLUSION 160
7.0 REFERENCES 162

TABLE OF FIGURES

Figure 1: A high level block diagram of a typical 4D/RCS reference model architecture.. 19
Figure 2. The fundamental structure of a 4D/RCS control loop. 21
Figure 3. The basic internal structure of a 4D/RCS control loop..... 23
Figure 4. Internal structure of a RCS_NODE.....28
Figure 5. A 4D/RCS reference model architecture for an individual vehicle.. 29
Figure 6. Three views of the 4D/RCS architecture..... 42
Figure 7. Five levels of the 4D/RCS architecture for Demo III. 43
Figure 8. The command and plan structure for Demo III. 50
Figure 9. A typical 4D/RCS computational node, RCS_NODE. 59
Figure 10. Internal structure of Behavior Generation (BG) processes 61
Figure 11. Two forms of plan representations. 69
Figure 12. A task plan for three agents. 70
Figure 13. A diagram of planning operations within a 4D/RCS node.. 71
Figure 14. The inner structure of the Executor (EX) subprocess. 74
Figure 15. A 4D/RCS timing diagram.. 78
Figure 16. Replanning at T(i)/10 intervals. 79
Figure 17. World Modeling (WM) and Value Judgment (VJ) processes 84
Figure 18. The structure of a typical entity frame.. 90
Figure 19. An entity frame. 92
Figure 20. An entity prototype 96
Figure 21. Attribute, entity, class, and value images. 98
Figure 22. Relationship between entity images and entity frames..... 100
Figure 23. A sensor egosphere for a camera.. 102
Figure 24. Two views of a velocity egosphere..... 103
Figure 25. A 2-D top down projection of four egosphere representations..... 104
Figure 26. A Mercator projection of the head egosphere..... 105
Figure 27. The structure of a typical event frame. 109
Figure 28. Grouping of subevents into events along the time line..... 112
Figure 29. A timing diagram for 4D/RCS..... 113
Figure 30. Data structures in immediate experience, short term, and long term memory. 117
Figure 31. Pointers between entity images and entity frames..... 123
Figure 32. The network of relationships between images and entity frames 124
Figure 33. Image processing functions and data flow at a typical level in the SP hierarchy..... 127
Figure 34. Relationships within a typical node of the 4D/RCS architecture 129
Figure 35. Interaction between sensory processing (SP) and world modeling (WM). 131
Figure 36. Basic scheme for 4-D image sequence understanding..... 133
Figure 37. A combination of the 4-D approach with RCS [redrawn from Dickmanns 96]. 134
Figure 38. Another view of the 4-D approach [from Dickmanns 96]..... 136
Figure 39. The 4-D recursive estimation concept integrated with a generic RCS BG process. 137
Figure 40. The 4D/RCS for a generic level..... 138
Figure 41. Image processing at level 1..... 143
Figure 42. Image processing at level 2..... 149
Figure 43. An example of the 4D/RCS Generic Shell. 157

Figure 44. Generic shell BG modules at the Subsystem, Primitive, and Servo levels..... 159

LIST OF ACRONYMS

ADL	Architectural Description Language
AI	Artificial Intelligence
ARL	Army Research Laboratory
AUV	Autonomous Underwater Vehicle
BG	Behavior Generation
C3	Command, Control, and Communications
C4ISR	Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance
CCD	Charge Coupled Device
Df.	Definition
EX	Executor
FCS	Future Combat Systems
FLIR	Forward Looking Infrared Imaging
GPS	Global Positioning System
HQ	Headquarters
HMMWV	High Mobility Multipurpose Wheeled Vehicle
h	hour
IFF	Identification Friend-or-Foe
ISD	NIST Intelligent Systems Division
IPT	Integrated Product Teams
JA	Job assignor
JAUS	Joint Architecture for Unmanned Systems
JAUGS	Joint Architecture for Unmanned Ground Systems
JTA	Joint Technical Architecture
JTA-A	Joint Technical Architecture – Army
KD	Knowledge Database
kN	kilonewton
LADAR	Laser Radar
m	meter
ms	millisecond
min	minute
NIST	National Institute of Standards and Technology
OE	Operating Environment
OI	Operator Interface
OSD	the Office of the Secretary of Defense
PL	Planner
RCS	Real-time Control System
RSTA	Reconnaissance, Surveillance, and Target Acquisition
s	second
SC	Scheduler
SP	Sensory Processing
TACOM	U.S. Army Tank-Automotive and Armaments Command
TARDEC	Tank Automotive Research, Development and Engineering Center

TRM	Technical Reference Model
UML	Unified Modeling Language
UARV	Unmanned Armed Reconnaissance Vehicle
UAV	Unmanned Aerial Vehicle
UGS	Unattended Ground Sensors
UGV	Unmanned Ground Vehicle
UVA	Unmanned Vehicle Architecture
VRA	Vehicle Electronics (Vetronics) Reference Architecture
VJ	Value judgement
WM	world modeling
WSTAWG	Weapon System Technical Architecture Working Group
XUV	eXperimental Unmanned Vehicles

1.0 INTRODUCTION

4D/RCS is a methodology for conceptualizing, designing, engineering, integrating, and testing intelligent systems software for vehicle systems with any degree of autonomy, from manually operated to fully autonomous. The theoretical foundation for this methodology is the 4D/RCS reference model architecture.

Df. A **methodology** *is a set of methods, rules, and postulates employed by a discipline.*

Df. An **architecture** is *the structure that identifies, defines, and organizes components, their relationships, and principles of design; the assignment of functions to subsystems and the specification of the interfaces between subsystems.*

Df. A **reference model architecture** is *an architecture in which the entire collection of entities, relationships, and information units involved in interactions between and within subsystems and components are defined and modeled.*

Df. An **intelligent system** is *a system with the ability to act appropriately in an uncertain environment.*

Df. An **appropriate action** is *that which maximizes the probability of successfully achieving the mission goals.*

Df. A **mission goal** is *a desired result that a mission is designed to achieve or maintain.*

Df. A **result** is *represented as a state or some integral measure of a state-time history.*

Df. A **mission** is *the highest level task assigned to the system.*

The 4D/RCS reference model architecture has the following properties:

1. It defines the functional elements, subsystems, interfaces, entities, relationships, and information units involved in intelligent vehicle systems.
2. It supports the selection of goals, the establishment of priorities and rules of engagement, the generation of plans, the decomposition of tasks, and the scheduling of activities; and it provides for feedback to be incorporated into control processes so that both deliberative and reactive behaviors can be combined into a single integrated system.
3. It supports the processing of signals from sensors into knowledge[1] of situations and relationships; and it provides for the storage of knowledge in representational forms that can support reasoning, decision-making, and intelligent control.

[1] Some researchers prefer a clear distinction between the terms information and knowledge. 4D/RCS, being a rich architecture, supports both and adopts the concept of a smooth transition between information and knowledge.

4. It provides both static (typically for long-term) and dynamic (typically for short-term) means for representing the richness and abundance of knowledge necessary to describe the environment and state of a battlefield and the intelligent vehicle systems operating within it.
5. It supports the transformation of information from sensor signals into symbolic and iconic representations of objects, events, and situations, including semantic, pragmatic, and causal relationships; and it supports transformations from iconic (pictorial) to descriptive (symbolic) forms, and vice versa.
6. It supports the acquisition (or learning) of new information and the integration and consolidation of newly acquired knowledge into long-term memory.
7. It provides for the representation of values, the computation of costs and benefits, the assessment of uncertainty and risk, the evaluation of plans and behavioral results, and the optimization of control laws.

Details of these properties will be given in the corresponding sections later in this document.

Standards Orientation

4D/RCS draws on a number of commercial and military standards to facilitate the design, development, debugging, maintenance, and upgrading of subsystems and software. 4D/RCS is compatible with all relevant standards developed under the following U.S. Department of Defense programs: the DoD Joint Technical Architecture (JTA) [DoD 02] and Technical Reference Model (TRM) [TRM 02], Battlefield Digitization [PEO C3T 02], the C4ISR Architecture Framework [CISA 02], the Army Joint Technical Architecture-Army [JTA-A 02] and Weapon System Technical Architecture Working Group (WSTAWG) [WSTAWG 02], the Office of the Secretary of Defense (OSD) Joint Architecture for Unmanned Ground Systems (JAUGS) [JAUGS 02], and the TARDEC Vehicle Electronics (Vetronics) Reference Architecture (VRA) [Vetronics 02, VRA 01].

JTA and JTA-A define an architecture to have three interrelated views, operational architecture, technical architecture, and systems architecture. The 4D/RCS reference model architecture specifies tasks, commands, and their planning and execution, organizational units, and information flows that support the operational architecture. 4D/RCS provides a development and application framework for the standards and conventions that the technical architecture covers. The 4D/RCS control hierarchy provides a logical layout that supports the system architecture. These demonstrate the comprehensiveness of 4D/RCS, covering all the JTA and JTA-A architectural aspects.

4D/RCS provides a methodology by which military systems that meet the operational requirements defined in the JAUGS Domain Model can be engineered to meet the performance specifications defined in the JAUGS Reference Architecture. 4D/RCS has been included in the current draft TARDEC VRA as a mandate for robotic systems. 4D/RCS endorses VRA's assessment that the VRA and 4D/RCS development teams will monitor each other's progress

and continue to interact on a regular basis. 4D/RCS is contributing to the various Integrated Product Teams (IPT) in WSTAWG.

4D/RCS plans to continue contributing to major DoD standards organizations and fostering an open system based architectural environment. Such an environment facilitates major DoD goals of interoperability and time/cost reduction for system development. An open system based architectural environment also facilitates the Army's goal of transitioning to the Objective Force.

Science and Technology Orientation

4D/RCS integrates the NIST Real-time Control System (RCS) [Albus and Meystel 96] architecture with the German (Universitat der Bundeswehr Munchen) VaMoRs 4-D approach to dynamic machine vision [Dickmanns, et al. 94]. It incorporates many concepts developed under the U.S. Department of Defense Demo I, Demo II, and Demo III programs [Albus et al. 02, Shoemaker et al. 99, Shoemaker et al. 98, Haas, et al. 96], which demonstrated increasing levels of robotic vehicle autonomy. The theory embodied in 4D/RCS borrows heavily from cognitive psychology, semiotics, neuroscience, and artificial intelligence. It incorporates concepts and techniques from control theory, operations research, game theory, pattern recognition, image understanding, automata theory, and cybernetics from the application domain perspective. The 4D/RCS architecture consists of a multi-resolution hierarchy of feedback control loops between sensing and acting that integrate reactive behavior with perception[2], world modeling, and planning – and forming a hybrid deliberative/reactive system [Rasmussen 02, 01, Maximov 92]. A review of projects that have used RCS and a description of how RCS relates to other intelligent system architectures are contained in [Albus and Meystel 01] and [Meystel and Albus 02].

4D/RCS also adopts many software engineering techniques, including object-orientation [Huang et al. 01, Huang and Messina 96], reuse, interoperability [Gazi et al. 01], component-based software [Horst 00], software specification [Messina and Huang 02], testing, and formal models [Dabrowski 99].

Domains

Versions 0.1 and 1.0 of the 4D/RCS architecture were designed to support the design, engineering, integration, and test of intelligent system software for experimental unmanned ground vehicles developed under the Army Research Laboratory Demo III program [Shoemaker et al 99, Bornstein 02, Hong et al. 02a-d, Murphy et al. 02]. Version 2.0 of the 4D/RCS reference model architecture is intended to facilitate the integration of a wide variety of unmanned and manned vehicles and sensors (ground, air, and amphibious) into an effective fighting force system-of-systems within the framework of the Army Future Combat Systems (FCS).

[2] 4D/RCS terms. Will be defined in the document.

1.1 Scope

The 4D/RCS methodology addresses the problem of intelligent control at three layers of abstraction: (1) a conceptual framework; (2) a reference model architecture; and (3) engineering guidelines. They are a series of successively refined descriptions in terms of the details and specific features; in other words, the upper layers mold the lower. The conceptual framework provides the overall shape for the reference model architecture. The reference model architecture provides guidance for how to apply the engineering guidelines.

If, during implementation, the engineering guidelines require changing, it will be desirable to do so without changing the reference model architecture, since adherence to the reference model helps ensure that the engineering guidelines are compatible with one another. The system builder must understand the reference model in order to apply it.

1.1.1 A Conceptual Framework

The 4D/RCS architecture is derived from the proven RCS architecture. RCS is domain independent. The 4D/RCS conceptual framework is a mapping of RCS tenets to the domain of intelligent vehicle systems for military use.

At the highest layer of abstraction, 4D/RCS is intended to provide a conceptual framework for addressing the general problem of intelligent vehicle systems, operating in man-made and natural environments, pursuing mission goals, and supervised by human commanders.

The 4D/RCS conceptual framework spans the entire range of operations that affect intelligent vehicles, from those that take place over time periods of milliseconds and distances of millimeters to those that take place over time periods of months and distances of thousands of kilometers. The 4D/RCS model is intended to allow for the representation of activities that range from detailed dynamic analysis of a single actuator in a single vehicle subsystem to the combined activity of planning and control for hundreds of vehicles and human beings in full dimensional operations covering an entire theater of battle. In order to span the wide range of activities included within the conceptual framework, the 4D/RCS adopts a multilevel hierarchical architecture with different range and resolution in time and space at each level[3]. The 4D/RCS architecture design integrates easily into the information intensive structure of Future Combat Systems (FCS) [Boeing 02] and other advanced concepts for the armed forces of the United States in the 21st century. For example, a current implementation of 4D/RCS is being interfaced to a distributed, interactive combat simulation within the OneSAF simulation/training system [OTB 02]. This enables real and/or virtual vehicles controlled by the 4D/RCS architecture to participate in force-on-force exercises with tens or hundreds of real or virtual vehicles, some

[3] The term level is used interchangeably with "4D/RCS hierarchical control level" throughout this document, unless explicitly stated otherwise.

manned, some controlled by 4D/RCS real-time controllers, and others controlled by OneSAF autonomous behaviors.

4D/RCS provides a progressive framework in terms of human involvement. The full range of operations conforming to 4D/RCS can be either totally performed by soldiers or fully autonomous with human supervision.

1.1.2 A Reference Model Architecture

At a lower layer of abstraction, the 4D/RCS is a reference model architecture. The architecture applies sound hierarchical control principles and, at the same time, closely follows the existing command and control structure of the military hierarchy in assigning duties and responsibilities and in requiring knowledge, skills, and abilities.

The 4D/RCS defines functional processes at each hierarchical control level such that each process embodies a set of responsibilities and priorities that are typical of operational units in a military organization. This enables the 4D/RCS architecture to map directly onto the military command and control organization to which the intelligent vehicles are assigned. The result is a system architecture that is understandable and intuitive for the human commander and integrates easily into battle space visualization and simulation systems.

1.1.3. Engineering Guidelines

At a still lower layer of abstraction, the 4D/RCS is intended to provide engineering guidelines for building and testing specific intelligent vehicle systems. In order to build a practical system in the near term, 4D/RCS engineering guidelines have been developed bottom-up, starting with a single vehicle and its subsystems. The 4D/RCS engineering guidelines define how intelligent vehicles should be configured to work together in groups with other intelligent vehicles, both manned and unmanned, in units of various sizes.

The type of problems to be addressed by the 4D/RCS engineering guidelines include:

- Navigating and driving both on and off roads,
- Responding to human supervisor commands and requests,
- Accomplishing mission goals and priorities amid the uncertainties of the battlefield,
- Cooperating with friendly agents in conducting tactical behaviors,
- Acting appropriately with respect to hostile agents, and
- Reacting quickly, effectively, and resourcefully to obstacles and unexpected events.

Intelligent vehicle systems typically consist of a variety of sensors, actuators, navigation and driving systems, communications systems, mission packages, and weapons systems controlled by an intelligent controller.

Sensors may include charge coupled device (CCD) television cameras, laser radar (LADAR) range imaging cameras, forward looking infrared (FLIR) cameras, radar, acoustic arrays, chemical or radiation detectors, and radio frequency receivers, as well as inertial, force,

pressure, and temperature sensors. Actuators may include motors, pistons, steering, brakes, throttle, lasers, radio frequency (R.F.) transmitters, and pointing systems for cameras, antennas, and weapons.

Navigation and driving systems may include computer aided mission planning systems, digital terrain map management systems, route planning systems, GPS and inertial guidance systems, and machine vision systems for on and off road driving, with obstacle avoidance, target tracking, object classification, and image understanding capabilities.

Communications may include microwave, packet radio, and lasers, and may include relay via other ground vehicles, air vehicles, or satellite communications. Updates for terrain data or over-the-hill visualization may be supplied by unmanned air vehicles or satellite surveillance systems.

Mission packages may include reconnaissance, surveillance, and target acquisition systems, range finders, laser designators, direct or indirect fire weapons, counter mine equipment, sniper detection equipment, smoke generators, electronic warfare equipment, logistics support, or communications relay antennas. The weapons may include machine guns, recoilless rifles, cannons, missile launching systems, mortars, or a variety of non-lethal weapons.

It is, therefore, an extremely challenging task to engineer these intelligent vehicle systems. The 4D/RCS Engineering Guidelines address key software engineering issues that facilitate the intelligent vehicle systems lifecycle processes, including:

Software Reuse

Reuse reduces system development costs. The 4D/RCS engineering guidelines can be leveraged in software reuse at several levels of detail.

Minimally, the definition of an RCS_Node provides control software designers with a framework for attacking the problem. The hierarchical decomposition guidelines help designers decide on how to decompose the problem and assign responsibilities to software modules. This could be considered reuse of software design and has a flavor akin to the use of architectural patterns [Gamma 1994]. Merely following the textual descriptions of what the main sub-components of an RCS_Node are, their responsibilities, and interrelationships provides initial assistance to designers and developers.

NIST and other organizations have been investigating means of enabling reuse of portions of 4D/RCS code. A Generic shell (see Section 5) provides generic computing models for 4D/RCS-based hierarchical control systems, including interfacing and communication. Developers apply the generic building blocks to all the control nodes and interfaces to build a skeleton for their control systems. The developers then embed the application-specific behavioral or processing algorithms into the skeleton and fill in the interprocess interface templates.

A number of software engineering tools have been developed for constructing generic shell modules. These range in degree of formality from C++ templates to Unified Modeling Language (UML) and Architectural Description Languages (ADL) [Huang 2001] [Messina 2000] [Dabrowski 1999]. A.J. Barbera and M.L Fitzgerald have applied a task decomposition based RCS approach to many industrial automatic systems and have developed a variety of tools for visualization of control system execution. [ATR 02]. A RCS tool developed by John Horst at NIST uses Control Shell [Horst 00]. Another RCS tool developed by Will Shackleford at NIST is written in Java. [Gazi01] Additional tools are being developed at Ohio State University using a LabView front end, [Gazi01] and by Pathway Technologies using MatLab as a front end [Anathakrishnan02][4]

Tools that provide support for building, testing, and evaluating 4D/RCS-based systems are also being experimentally created [Balakirsky 2002]. The architect could use tools to create the initial specification for the system (hierarchy, timings, interfaces, etc.) and create the generic shell versions for the RCS_Nodes. A repository of 4D/RCS-compliant software components would then be available for developers to assemble directly or augment or customize. The tools would provide graphical and interactive means of building a system that is compliant with the reference architecture and that encourages (or enforces) reuse.

Interoperability and open systems

Complex, intelligent vehicle systems typically involve software components contributed by multiple development teams. Component and interface standards are crucial to system integration. 4D/RCS specifies nodes in terms of their functionality and interfaces. These present a comprehensive framework for developing and applying the standards, which, in turn, present open systems for components and subsystems to be easily integrated.

Large teams of software developers will have to work together. As technology evolves and subsystems improve, it is desirable to be able to directly upgrade components within the overall architecture to take advantage of improved capabilities. Therefore, an architecture such as 4D/RCS is essential for defining the software decomposition and interfaces that enable distributed development and component upgrades.

[4] Commercial equipment and materials are identified in order to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

2.0 A CONCEPTUAL FRAMEWORK

The 4D/RCS conceptual framework demonstrates how intelligent unmanned vehicle systems can be integrated into any military command and control structure.

Df. A **framework** is *a description of the functional elements, the representation of knowledge, and the flow of information within the system.*

4D/RCS integrates the functional elements, knowledge representations, and flow of information so that intelligent systems can analyze the past, perceive the present, and plan for the future. It enables systems to assess the cost, risk, and benefit of past events and future plans, and to make intelligent choices among alternative courses of action.

The 4D/RCS is a hybrid architecture, with both deliberative (reasoning and planning) and reactive (rapid response to exigencies) capabilities. At every level of the control hierarchy there are deliberative planning processes that receive goals and priorities from superiors and decompose them into subgoals and priorities for subordinates at levels below. At every level, reactive loops respond quickly to feedback to modify planned actions so that goals are accomplished despite unexpected events. Thus, planning and decision making are distributed throughout the hierarchy. At every level, plans are formulated, decisions are made, and reactive actions are taken locally by the units that are most affected and best able to analyze the situation and respond effectively.

At every level, sensory processing filters and processes information derived from observations by subordinate levels. Events are detected, objects recognized, situations analyzed, and status reported to superiors at the next higher level.

At every level, sensory processing and behavior generation processes have access to a model of the world that is resident in a knowledge database. This world model enables the intelligent system to analyze the past, plan for the future, and perceive sensory information in the context of expectations.

At every level, a set of cost functions enable value judgments and determine priorities that support intelligent decision making, planning, and situation analysis. This provides a robust form of value driven behavior that can function effectively in an environment filled with uncertainties and unpredictable events. It assures that intelligent vehicle systems will always obey direct orders from human commanders, and when out of contact, will respect the goals, priorities, and rules of engagement set by human commanders.

The 4D/RCS architecture maps naturally onto the military command and control structure. At the top level of any military hierarchy, there is a human commander supported by a staff that provides intelligence and decision support functions. This is where high-level strategy is defined and strategic goals are established. The top level commander decides what kind of

operations will be conducted, what rules of engagement will be followed, and what values will determine priorities and shape tactical decisions.

Throughout the hierarchy, strategic goals are decomposed through a chain of command that consists of operational units made up of intelligent agents (humans or machines), each of which possesses a particular combination of knowledge, skills, and abilities, and each of which has a well-defined set of duties and responsibilities. Each operational unit accepts tasks from a higher level unit and issues sub-tasks to subordinate units. Within each operational unit, intelligent agents are given job assignments and allocated resources with which to carry out their assignments; the intelligent agents then schedule their activities so as to achieve the goals of the jobs assigned to them. Each agent is expected to make local executive decisions to achieve goals on schedule by solving local problems and compensating for local unexpected events. Within a unit, each agent acts as a member of a team in planning and coordinating with peers at the same level, while simultaneously acting as the commander of a subordinate unit at the next lower level.

Each agent, within each operational unit, has knowledge of the world environment in which it must function. This knowledge includes state variables, maps, images, and symbolic descriptions of the state of the world. It also includes knowledge of objects and groups that exist in the environment, including their attributes and relationships, and knowledge of events and processes which develop over time. Knowledge is kept current and accurate through sensors and sensory processing systems that detect events and compute attributes of objects and situations in the world. Knowledge of the world also includes laws of nature that describe how the environment behaves under various conditions, as well as values and cost functions that can be used to evaluate the state of the world and the performance of the intelligent control system itself.

At the bottom of the hierarchy, the system performs physical actions (e.g., the movement of effectors such as wheels, tracks, arms, legs, thrusters, or control surfaces), which affect the environment. Simultaneously, sensors measure phenomena – including the effects of the system itself – in the environment. This process is a continuous loop of the environment affecting the robotic system and the robotic system affecting the environment.

For any chain of command, an organizational chart can be constructed that describes functional groupings and defines who reports to whom. However, organizational charts typically do not show all the communication pathways by which information flows throughout the organization. In particular, much information flows horizontally between agents and operational units, through both formal and informal channels. Multiple agents within operational units share knowledge about objects and events in the world, and status of other agents. For example, agents operating on the battlefield often can see each other and may respond to requests for help from peers without explicit orders from superiors. Also, plans developed in one operational unit may be communicated to other units for implementation.

4D/RCS explicitly allows for the exchange of information between organizational units and agents at the same level or different levels. Commands and status reports flow only between

supervisor and subordinates, but queries, replies, requests, and broadcasting of information – by posting in common memory or by messaging mechanisms – may be used to convey information between any of the units or agents in the entire 4D/RCS architecture.

The 4D/RCS organizational chain of command is defined by the duties and responsibilities of the various organizational units and agents and by the flow of commands and status reports between them, not by access to information or the ability to communicate. This means that while the relationships between supervisors and subordinates is in the form of a *tree*, the exchange of information between units and agents is a *graph* that, in principle, could be fully connected. In practice, however, the communication network is typically not fully connected because many of the units and agents simply have nothing to say to each other.

4D/RCS also explicitly allows the organizational hierarchy to be dynamically reconfigured in real-time during operation. This permits the organizational chain of command to change over time as units are added or removed from the chain of command, or moved from one place to another in the organization. For example, a wing of unmanned aircraft may be under the control of a base flight controller during take-off or landing, be handed off to a route flight controller, and be transferred to a combat engagement controller during attack and battle damage assessment. Similarly, unmanned ground vehicles may be transferred from one chain of command to another as conditions change on the battlefield. Individual agents may be promoted or replaced, and units reconfigured during combat operations to compensate for casualties.

3.0 4D/RCS REFERENCE MODEL ARCHITECTURE

A reference model architecture is the core of 4D/RCS. The 4D/RCS reference model architecture is characterized by a generic control node that is applied to all the hierarchical control levels. The node features specific functions, interfaces, information structures, and processing paradigms that enable intelligent behavior. The 4D/RCS hierarchical levels are scalable to facilitate systems of any degree of complexity.

3.1 The 4D/RCS Hierarchy

Figure 1 shows a high level block diagram of a 4D/RCS reference model architecture for a notional Future Combat System (FCS) battalion. 4D/RCS prescribes a hierarchical control principle that decomposed high level commands into actions that employ physical actuators and sensors. Characteristics such as timing and node functionality may differ in various implementations.

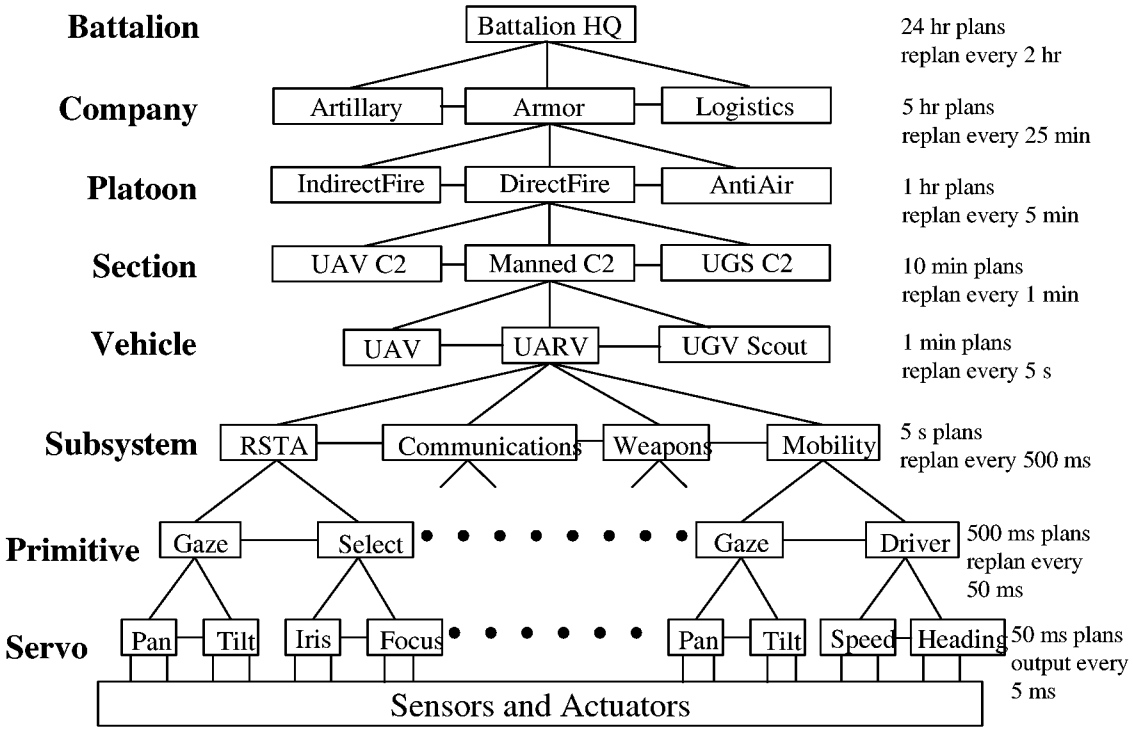


Figure 1: A high level block diagram of a typical 4D/RCS reference model architecture. Commands flow down the hierarchy, and status feedback and sensory information flows up. Large amounts of communication may occur between nodes at the same level, particularly within the same subtree of the command tree. UAV = Unmanned Air Vehicle, UARV = Unmanned Armed Reconnissance Vehicle, UGS = Unattended Ground Sensors

At the Servo level, commands to actuator groups are decomposed into control signals to individual actuators. In the example shown in Figure 1, outputs to actuators are generated every 5 milliseconds (ms). Plans that look ahead 50 ms are regenerated for each actuator every 5 ms. Plans of individual actuators are synchronized so that coordinated motion can be achieved for multiple actuators within an actuator group.

At the Primitive level, multiple actuator groups are coordinated and dynamical interactions between actuator groups are taken into account. Plans look ahead 500 ms and are recomputed every 50 ms.

At the Subsystem level, all the components within an entire subsystem are coordinated, and planning takes into consideration issues such as obstacle avoidance and gaze control. Plans look ahead 5 seconds (s) and replanning occurs every 500 ms.

At the Vehicle level, all the subsystems within an entire vehicle are coordinated to generate tactical behaviors. Plans look ahead 1 min and replanning occurs every 5 s.

At the Section level, multiple vehicles are coordinated to generate joint tactical behaviors. Plans look ahead about 10 minutes (min) and replanning occurs about every minute.

At the Platoon level, multiple sections containing a total of 10 or more vehicles of different types are coordinated to generate platoon tactics. Plans look ahead about an hour (hr) and replanning occurs about every 5 min.

At the Company level, multiple platoons containing a total of 40 or more vehicles of different types are coordinated to generate company tactics. Plans look ahead about 5 hr and replanning occurs about every 25 min.

At the Battalion level, multiple companies containing a total of 160 or more vehicles of different types are coordinated to generate battalion tactics. Plans look ahead about 24 hr and replanning occurs at least every 2 hours.

At all levels, task commands are decomposed into jobs for lower level units and coordinated schedules for subordinates are generated. At all levels, communication between peers enables coordinated actions. At all levels, feedback from lower levels is used to cycle subtasks and to compensate for deviations from the planned situations.

Df. A **task command** is *a command to a BG process to do a task, or to modify an ongoing task.*

Each node within the hierarchy shown in Figure 1 functions as a goal-driven, model-based, closed-loop controller. Each node is capable of accepting and decomposing task commands with goals into actions that accomplish task goals despite unexpected conditions and dynamic perturbations in the world. At the heart of the control loop through each node is a rich, dynamic world model that provides the node with an internal model of the external world. This is illustrated in Figure 2. In each node, the world model provides a site for data fusion, acts as a

buffer between perception and behavior, and supports both sensory processing and behavior generation.

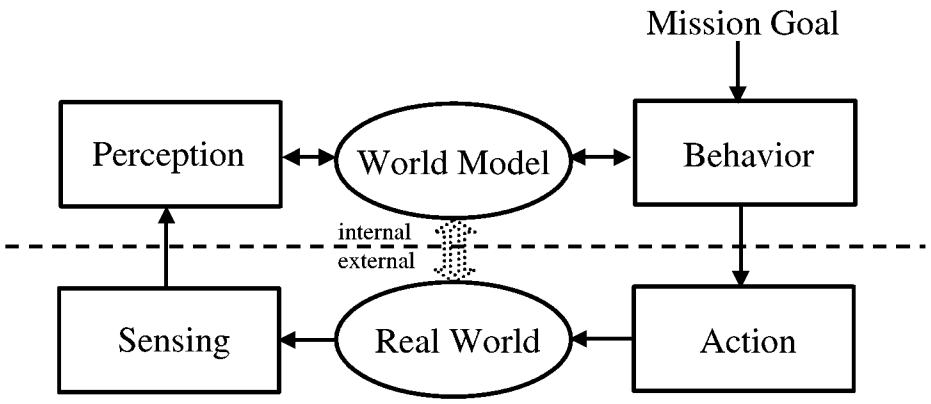


Figure 2. The fundamental structure of a 4D/RCS control loop. An internal world model of the external world provides support to both perception and behavior. Sensors measure properties of the external world. Perception extracts the information necessary to keep the world model current and accurate from the sensory data stream. Behavior uses the world model to decompose goals into appropriate action.

In support of behavior generation, the world model provides knowledge of the environment with range and resolution in space and time that is appropriate to task decomposition and control decisions that are the responsibility of that node. The world model also provides simulation and modeling for planning and reasoning about the future. For example, the world model can simulate results of hypothesized actions that can be evaluated and compared with the current state of the world. This enables behavior generation to plan and control actions that are most likely to achieve the given goal at minimum cost and maximum benefit.

Df. Task decomposition is *a process by which a task given to a BG process at one level is decomposed into a set of sequences of subtasks to be given to a set of subordinate BG processes at the next lower level.*

Df. Planning is *a process of generating and/or selecting a plan to accomplish a task or job*

Df. A **state** is the *dynamic condition of an entity or a process at a point in time.*

In support of sensory processing, the world model provides predictions that can be matched with observations for recursive estimation and Kalman filtering [Kalman 02]. The world model also provides hypotheses for gestalt grouping and segmentation. Thus, each node in the 4D/RCS hierarchy is an intelligent system that accepts goals from above and generates commands for subordinates so as to achieve those goals.

The centrality of the world model to each control loop is a principal distinguishing feature between 4D/RCS and behaviorist (i.e., purely reactive) architectures [Brooks 91, 86]. Behaviorist architectures rely solely on sensory feedback from the world. They do not fuse information from multiple sensors over time, nor do they integrate sensory feedback with a priori knowledge. All behavior is a reaction to immediate sensory feedback. In contrast, the 4D/RCS world model integrates all available knowledge into an internal representation that is far richer and more complete than is available from immediate sensory feedback alone. This enables more sophisticated behavior than can be achieved from purely reactive systems.

The character and structure of the world model also distinguishes 4D/RCS from conventional artificial intelligence (AI) architectures. Most AI world models are purely symbolic. In 4D/RCS, the world model is much more than a symbolic abstraction of the world. It is, rather, a combination of instantaneous signal values from sensors, state variables, images, and maps that are linked to symbolic representations of entities, events, objects, classes, situations, and relationships in a composite of immediate experience, short-term memory, and long-term memory.

Df. A **map** is a *two-dimensional array of attributes and entities that are scaled to, and registered with, known locations in the world.*

A high level diagram of the internal structure of the world model and value judgment system is illustrated in Figure 3. Within the knowledge database[2], iconic information in the form of images and maps are linked to each other and to symbolic information in the form of entities and events. Situations and relationships between and entities, events, images, and maps are represented by *pointers*. Pointers that link symbolic data structures to each other form syntactic, semantic, causal, and situational networks. Pointers that link symbolic data structures to regions in images and maps provide *symbol grounding* and enable the world model to project its *understanding* of reality onto the physical world. A *world modeling* process maintains the knowledge database and uses information stored in the knowledge database to generate predictions for sensory processing and simulations for behavior generation. Predictions are compared with observations and errors are used to generate updates for the knowledge database. Simulations of tentative plans are evaluated by value judgment to select the "best" plan for execution.

Df. A **plan** is a *set of subtasks and subgoals that form a path from the starting state to the goal state.*

As suggested in Figure 3, the 4D/RCS control loop contains four functional elements: sensory processing, world modeling, value judgment, and behavior generation.

Df. **Functional elements** are the *fundamental computational processes from which the system is composed.*

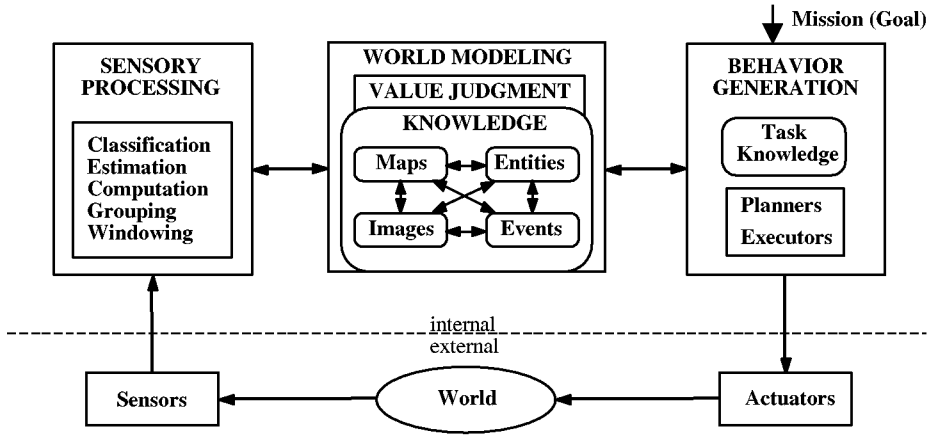


Figure 3. The basic internal structure of a 4D/RCS control loop. Sensory processing performs the functions of windowing, grouping, computation, estimation, and classification on input from sensors. World modeling maintains knowledge in the form of images, maps, entities, and events with states, attributes, and values. Relationships between images, maps, entities, and events are defined by pointers. These relationships include class membership, ontologies, situations, and inheritance. Value judgment provides criteria for decision making. Behavior generation is responsible for planning and execution of behaviors.

3.2 Sensory Processing

Df. Sensory Processing is a *set of processes that operate on sensor signals to detect, measure, and classify entities and events and derive useful information about the world.*

Sensory processing performs the operations of windowing, grouping, computation, filtering, and classification, or recognition at many different levels. Sensory processing computes observed features and attributes, and compares them with predictions from internal models. Correlation between sensed observations and internally generated expectations are used to detect events and recognize entities and situations. Differences between sensed observations and internally generated predictions are used to update internal models. Perception results when the internal world model matches the external world.

Sensory processing accepts signals from sensors that measure properties of the external world or conditions internal to the system itself. In general, sensors do not directly measure the state of the world. Sensors only measure phenomena that result from the state of the world. Signals generated by sensors may be affected by control actions that cause the sensors to move through the world. Sensor signals are also corrupted by noise. The set of equations that describe how sensor signals depend on the state of the world, the control action, and sensor noise is called a *measurement model.*

A measurement model is typically of the form

$$\mathbf{y = H(x, u, \eta)}$$

where:

y = signals from sensors
x = state of the world
u = control action
η = sensor noise
H = a function that relates sensor signals to world state, control action, and noise

A linearized form of the measurement model is typically of the form:

$$\mathbf{y = Cx + Du + \eta}$$

where:

C is a matrix that defines how sensor signals depend on the world state
D is a matrix that defines how sensor signals depend on the control action

3.3 World Modeling

Df. World modeling is *a set of processes that construct and maintain a world model*

Df. A **world model** is *an internal representation of the world.*

World modeling is a process that performs four principal functions:

1. It maintains a knowledge database of images, maps, objects, agents, situations, relationships, and knowledge of task skills and laws of nature and relationships among them.

2. It maintains a best estimate of the state of the world to be used as the basis for predicting sensory feedback and planning future actions.

3. It predicts (possibly with several hypotheses) sensory observations based on the estimated state of the world. Predicted signals can be used by sensory processing to configure filters, masks, windows, and templates for correlation, model matching, and recursive estimation.

4. It simulates results of possible future plans based on the estimated state of the world and planned actions. Simulated results are evaluated by the value judgment system in order to select the best plan for execution.

The world model includes a knowledge database and set of world modeling processes. The world model includes models of portions of the environment, images, maps, models of entities, events, and agents, rules, task knowledge, abstract data structures, and pointers that represent relationships, and a system model that includes the intelligent system itself. The world

model knowledge database is dynamic, multiresolutional, and distributed over the 4D/RCS nodes. It is maintained in each node by a local world modeling process.

Df. A **system model** is *a set of differential equations (for a continuous system) or difference equations (for a discrete system) that predict how a system will respond to a given input.*

A **system model** is typically of the form

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}, \boldsymbol{\gamma})$$

where:

$\mathbf{x}$ = the state of the system
$\dot{\mathbf{x}}$ = the rate of change in the system state
$\mathbf{u}$ = the control action
$\boldsymbol{\gamma}$ = system noise
$\mathbf{f}$ = a function that defines how the system state changes over time in response to control actions

A **linearized form** of the above system model is of the form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} + \boldsymbol{\gamma}$$

where:

$\mathbf{A}$ is a matrix that defines how the system state evolves over time without control action.
$\mathbf{B}$ is a matrix that defines how the control action affects the system state.

Df. A **knowledge database** contains *the data structures and the static and dynamic information that together with world modeling processes form the intelligent system's world model.*

The knowledge database has three parts:

(1) **immediate experience** consisting of iconic representations in the form of current values of sensor signals, camera images, maps, etc. Immediate experience also consists of entities, events, pointers, and observed, estimated, and predicted attributes and state variables.

Df. **Iconic knowledge** is *2D array data in which the dimensions of the array correspond to dimensions in an image. The value of each element of the array may be boolean data or real number data representing a physical attribute such as light intensity, color, or terrain elevation.*

(2) **short-term memory** consisting of symbolic representations in the form of a list of entities that are the subject of current attention, pointers that define relationships and class membership, and queues of recent events at various levels of temporal resolution.

(3) **long-term memory** consisting of symbolic representations of all the objects, events, classes, relationships, and rules that are known to the intelligent system.

3.4 Value Judgment

Df. **Value judgment** *is* a process that *computes value, determines importance, assesses reliability, and generates reward and punishment.*

Value judgment evaluates perceived and planned situations, thereby enabling behavior generation to select goals and set priorities. It computes what is important (for attention), and what is rewarding or punishing (for learning). Value judgment assigns priorities and computes the level of resources to be allocated to tasks. It assigns values to recognized objects and events, and computes confidence factors for observed, estimated, and predicted attributes and states.

3.5 Behavior Generation

Df. **Behavior generation** is *planning and control of actions designed to achieve behavioral goals.*

Df. A **behavioral goal** is *a desired state that a behavior is designed to achieve or maintain*

Behavior generation plans and executes tasks in order to successfully accomplish mission goals. Behavior generation uses task knowledge, skills, and abilities along with knowledge in the world model to plan and control appropriate behavior in the pursuit of goals. Behavior generation accepts task commands with goals and priorities, formulates and/or selects plans, and controls action. Behavior generation develops or selects plans by using a priori task knowledge and value judgment functions combined with real-time information provided by world modeling to find the best assignment of tools and resources to agents, and to find the best schedule of actions (i.e., the most efficient plan to get from an anticipated starting state to a goal state). Behavior generation controls action by both feed forward actions and by feedback error compensation. Goals, feedback, and feed forward signals are combined in a control law.

Df. A **control law** is *a set of equations that computes control action given predicted state, desired state, and feed forward action.*

A **control law** is typically of the form

$$\mathbf{u = g(uff, xd, x^*)}$$

where:

u = control action

uff = feed forward control action (from a plan or inverse model)

xd = desired world state (from a command)
x* = predicted world state (from the world model)

A **linearized form** of a control law is

$$\mathbf{u = uff + G(x^* - xd)}$$

where:

G = a matrix that defines the feedback compensation applied to the difference between the desired and predicted state of the world

In simple cases, feed forward actions can be computed by applying a desired goal to an inverse model of the system. However, for complex systems, the world model typically has no inverse, and feed forward actions can only be discovered through planning. Planning typically involves a heuristic search through the space of possible actions using a world model to predict the results of those actions. A value judgment process is then invoked to evaluate potential plans and choose the best plan for execution.

3.6 The RCS_NODE

In the 4D/RCS reference architecture, behavior generation, world modeling, sensory processing, value judgment, and the knowledge database are organized into RCS_NODEs.

Df. A **RCS_NODE** is *an organizational unit of a 4D/RCS system that processes sensory information, computes values, maintains a world model, generates predictions, formulates plans, and executes tasks.*

Figure 4 illustrates the relationships within a single RCS_NODE of the 4D/RCS architecture. The interconnections between sensory processing, world modeling, and behavior generation close a reactive feedback control loop between sensory measurements and commanded action. The interconnections between behavior generation, world modeling, and value judgment enable deliberative planning and reasoning about future actions. The interconnections between sensory processing, world modeling, and value judgment enable knowledge acquisition, situation evaluation, and learning. Within sensory processing, observed input from sensors and lower level nodes is compared with predictions generated by world modeling. Differences between observations and predictions is used by world modeling to update the knowledge database. This can implement recursive estimation processes such as Kalman filtering. Within behavior generation, goals from higher levels are compared with the state of the world as estimated in the knowledge database. Behavior generation typically involve planning and execution functions. Differences between goals and estimated states are used to generate action. Information in the knowledge database of each node can be exchanged with peer nodes for purposes of synchronization and information sharing. Any or all of the processes within a node may communicate with an Operator Interface.

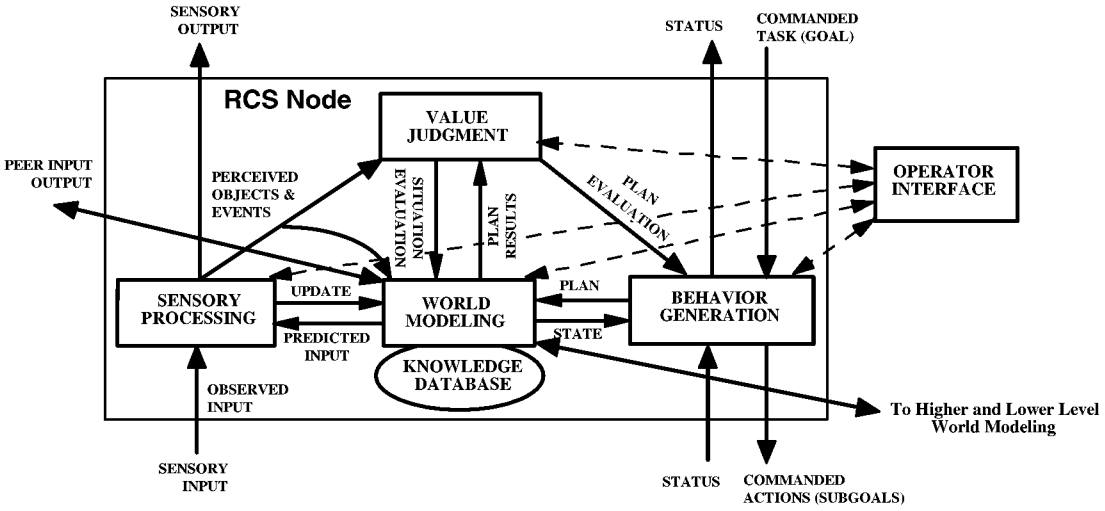


Figure 4. Internal structure of a RCS_NODE. The functional elements within a RCS_NODE are behavior generation, sensory processing, world modeling, and value judgment. These are supported by a knowledge database, and a communication system that interconnects the functional processes and the knowledge database. Each functional element in the node may have an operator interface. The connections to the Operator Interface enable a human operator to input commands, to override or modify system behavior, to perform various types of teleoperation, to switch control modes (e.g., automatic, teleoperation, single step, pause), and to observe the values of state variables, images, maps, and entity attributes. The Operator Interface can also be used for programming, debugging, and maintenance.

A RCS_NODE is analogous to Koestler's concept of a "holon" [Koestler 67]. Each RCS_NODE looks upward to a higher level node from which it takes commands, for which it provides sensory information, and to which it reports status. Each RCS_NODE also looks downward to one or more lower level nodes to which it issues commands, and from which it accepts sensory information and status. Each RCS_NODE may also communicate with peer nodes with which it exchanges information. A RCS_NODE is often abbreviated as a node in this document.

3.7 An Organization of RCS_NODEs

A collection of RCS_NODES can be used to construct a distributed hierarchical reference model architecture such as shown in Figure 5. Each node in the 4D/RCS architecture corresponds to a functional unit in a military command and control hierarchy. Depending on *where* the generic node resides in the hierarchy, it might serve as an intelligent controller for an actuator, a subsystem, a vehicle, a section, a platoon, a company, battalion, or higher level organizational unit. Each generic node (or a set of processes within a node) might either be implemented as an intelligent supervised-autonomy controller or be performed by a human or management unit at any level in the military command and control structure.

Df. Intelligent supervised-autonomy controllers are *controllers capable of accepting commands from human supervisors and executing those commands with little or no further input from humans in unstructured and often hostile environments.*

An intelligent, supervised-autonomy controller is intelligent in that it is capable of executing its assigned mission with or without direct communication from a human supervisor. It is supervised in that it responds to commands from superiors with discipline in response to established rules of engagement as would any well disciplined human soldier. It is autonomous in that it is capable of formulating plans and coordinating with other intelligent agents in the execution of mission assignments. Environments in which UGVs with supervised-autonomy controllers are required to operate include urban warfare zones, rural battlefields, mountains, woods, farmlands, or desert terrain, as well as all kinds of weather during day or night.

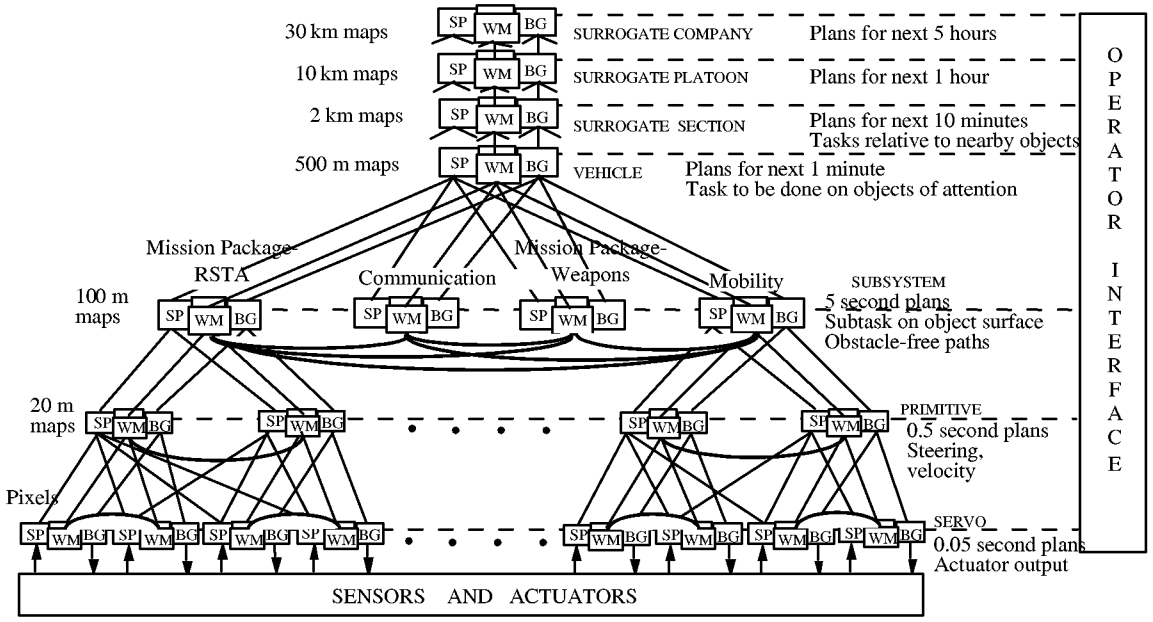


Figure 5. A 4D/RCS reference model architecture for an individual vehicle. Processing nodes, RCS_NODES, are organized such that the behavior generation (BG) processes form a command tree. Information in the knowledge database (KD) is shared between world modeling (WM) processes in nodes above, below, and at the same level within the same subtree. KD structures are not shown in this figure. On the right, are examples of the functional characteristics of the behavior generation (BG) processes at each level. On the left, are examples of the scale of maps generated by the sensory processing (SP) processes and populated by the WM in the KD knowledge database at each level. Sensory data paths flowing up the hierarchy typically form a graph, not a tree. Value judgment (VJ) processes are hidden behind WM processes. A control loop may be closed at every node. An operator interface may provide input to, and obtain output from, processes in every node.

In Figure 5, each node consists of a behavior generation (BG), world modeling (WM), and sensory processing (SP), and knowledge database (KD) (not shown in Figure 5). Most nodes also contain a value judgment (VJ) process (hidden behind the WM process in Figure 5). Each of the nodes can therefore function as an intelligent controller. An operator interface may access processes in all nodes at all levels.

Figure 5 illustrates a vehicle system with four subsystems: mobility, communication, and two mission packages. Each of the four subsystems has one or more mechanisms. Each of the mechanisms have one or more actuators and sensors. For example, the mobility subsystem may consist of a navigation and driving controller with several subordinate controllers for steering, braking, throttle, and gear shift, plus ignition, lights, horn, and turn signals, each of which has one or more actuators and sensors. The communication subsystem might consist of a message encoding subsystem, a protocol syntax generator, and communications bus interface, plus antenna pointing and band selection actuators. The vehicle control system should be able to incorporate a variety of modular mission packages, each of which may contain a number of sensors and actuators. For example, a weapons mission package might have loading, aiming, and firing subsystems each with a number of sensors and actuators. A reconnaissance, surveillance, and target acquisition (RSTA) mission package might consist of mechanisms that use cameras, LADARs, FLIRs, radar, and acoustic sensors to detect and track objects, surfaces, edges and points, and compute trajectories for laser range finders, or pan, tilt, and focus actuators.

The operator interface (OI) provides the capability for the operator to interact with the system at any time at a number of different levels – to adjust parameters, to change speed, to select or verify targets, or to authorize the use of weapons. The OI provides a means to insert commands, change missions, halt the system, alter priorities, perform identification friend-or-foe (IFF), or monitor any of the system functions. The OI can send commands or requests to any BG process, or display information from any SP, WM, or VJ process. It can display any of the state variables in the KD at a rate and latency dictated by the communications bandwidth. Using the OI, a human operator can view situational maps with topographic features and both friendly and enemy forces indicated with overlays. The operator may use the OI to generate graphics images of motion paths, or display control programs (plans) in advance, or while they are being executed. The OI may also provide a mechanism to run diagnostic programs in the case of system malfunctions.

In Figure 5, three levels of control are shown above the node representing the individual vehicle. These three additional levels represent a surrogate chain of command that, in principle, exists above the individual vehicle. Because each vehicle is semi-autonomous, it carries a copy of the control nodes that otherwise would exist in its superiors if those superiors were tightly coupled in an integrated control structure. Individual vehicles are physically separated, and may only occasionally be in contact with each other or with their superiors through a low bandwidth and often unreliable communication channel. It is necessary for each vehicle to carry a surrogate chain of command that performs the functions of its superiors in the command chain.

The surrogate chain of command serves four functions. First, it provides each vehicle with an estimate of what its superiors would command it to do if they were in direct communication. Second, it enables any vehicle to assume the duties of any of its superiors in the event this should become necessary. Third, it provides a natural interface for human commanders at the section, platoon, or company level to interface with the vehicle at a level relevant to the task being addressed. Fourth, it enables each vehicle to dedicate a separate node to handle each of the higher level tasks. In this example, the surrogate chain of command

consists of three levels with three different planning horizons (ten minutes, one hour, and five hours). These three levels deal with external objects and maps at three different scales and ranges. There, of course, may be more than three levels above the vehicle.

In Figure 5, the horizontal curved lines between WM processes represent the sharing of state information in the knowledge database between nodes within subtrees in order to synchronize related tasks. The vertical lines between WM processes represent the sharing of knowledge required to populate maps and abstract data structures, and to perform recursive estimation of state variables at various levels in the world model.

The functionality of each level in the 4D/RCS reference model hierarchy is defined by the functionality, characteristic timing, bandwidth, and algorithms chosen by BG processes for decomposing tasks and goals at each level. Typically these are design choices that depend on the dynamics of the processes being controlled. The numbers shown on the right in Figure 5 represent planning horizons appropriate for a vehicle. For other types of systems, different numerical values would be derived from design parameters. The scale of the maps on the left in Figure 5 indicates the range of the maps at that level in the world model. The number of pixels in the maps is typically constant; thus the resolution of the maps decreases at each higher level.

3.8 Hierarchical Levels

The complexity inherent in intelligent systems can be managed through partition into hierarchical levels. Intelligent systems are inherently complex. Hierarchical leveling is a common method for organizing complex systems that has been used in many different types of organizations throughout history for effectiveness and efficiency of command and control. In a hierarchical control system, higher level nodes have broader scope and longer time horizons with less concern for detail. Lower level nodes have narrower scope and shorter time horizons with more focus on detail. At no level does a node have to cope with both broad scope and high level of detail. This limits the responsibility and load for all the nodes at all levels and enables the design of systems of arbitrary complexity, without computational overload in any node and any level.

4D/RCS uses the principle of hierarchical leveling to facilitate software reuse. All the nodes in the 4D/RCS architecture have many features in common. These include basic read, write, decision-making, communications, timing, record keeping, and debugging features. Generic nodes that provide these common features can be used to define organizational units at all levels. Each specific node can then be customized for its specific functional responsibilities by embedding level- and node-specific algorithms and knowledge. In the 4D/RCS reference architecture, behavior generation processes at the upper levels in the hierarchy are customized to generate long-range strategic plans consisting of major milestones, while lower level behavior generation processes successively decompose the long-range plans into short-range tactical plans with detailed activity goals. Sensory processing functions are customized at lower levels to operate on data over local neighborhoods and short time intervals, while at higher levels they integrate data over long time intervals and large spatial regions. At low levels, the knowledge database is filled with short-term, short-range, fine-grained information, while at higher levels it

is filled with knowledge that is broad in scope and generalized over large regions of space and time. At every level, feedback loops are closed to provide reactive behavior, with high-bandwidth fast-response loops at lower levels, and slower more deliberative behavior at higher levels.

Hierarchical leveling enables optimal use of iconic memory in the representation of time and space. At each level, state variables, images, and maps are maintained to the resolution in space and time that is appropriate to that level. At each successively lower level in the hierarchy, as detail is geometrically increased, the range of computation is geometrically decreased. Also, as temporal resolution is increased, the span of interest decreases. This produces a ratio that remains relatively constant throughout the hierarchy. As a result, at each level, behavior generation functions make plans of roughly the same number of steps. At higher levels, the space of planning options is larger and world modeling simulations are more complex, but there is more time available between replanning intervals for planning processes to search for an acceptable or optimal plan. Thus, hierarchical leveling keeps the amount of computing resources needed for behavior generation in each node within manageable limits.

Also at each level, entities with a lower level of abstraction are grouped to form entities with a higher level of abstraction. The effect is to geometrically increase the scope and encapsulate the detail of entities and events observed in the world. Thus, at each level, sensory processing functions are responsible for entities that contain roughly the same number of sub-entities.

At each level, events with a lower level of abstraction are grouped to form events with a higher level of abstraction along the time line. Thus, short term memory events at lower levels are much more detailed than short-term memory events at higher levels, but the historical record in short-term memory at lower levels covers a shorter time frame than short-term memory at higher levels. Correspondingly, plans at higher levels are longer term and less detailed than plans at lower levels.

This kind of leveling is typical of the military command and control hierarchy. At the top, strategic objectives are chosen and priorities defined that influence the selection of goals and the prioritization of tasks throughout the entire hierarchy. The details of execution are left to subordinates.

At intermediate levels, tasks with goals and priorities are received from the level above, and sub tasks with sub goals and attention priorities are output to the level below. In the intelligent vehicle environment, intermediate level tasks might be of the form: <go to position at map coordinates x,y>, <advance in formation along line z>, <engage enemy units at time t>, etc. The details of execution are left to subordinates.

At each level in the 4D/RCS hierarchy, higher-level, more global tasks are decomposed and focused into concurrent strings of more narrow and finer resolution tasks. The effect of each hierarchical level is thus to geometrically refine the detail of the task and limit the view of the world, so as to keep computational loads within limits that can be handled by individual intelligent agents, such as 4D/RCS nodes or ordinary human beings.

3.9 Focus of Attention

In 4D/RCS systems, complexity of the real world environment can also be managed through focusing attention. Intelligent systems must operate in a real world environment that is rich with detail. The real world environment contains an infinite variety of real objects, such as the ground, rocks, grass, sand, mud, trees, bushes, buildings, posts, ravines, rivers, roads, enemy and friendly positions, vehicles, weapons, and people. The background may contain millions of leaves, twigs, and grains of sand. The environment also contains elements of nature, such as wind, rain, snow, sunlight, and darkness. All of these objects and elements have states and may cause, or be part of, events and situations. The environment also contains a practically infinite regression of detail, and the world itself extends indefinitely far in every direction.

Yet, the computational resources available to any intelligent system are finite. No matter how fast and powerful computers become, the amount of computational resources that can be embedded in any practical system will be limited. Therefore, it is imperative that the intelligent system focus the available computing resources on what is important, and ignore what is irrelevant. In each situation, the intelligent system should know what it does not know, and know whether is it important to find out. Of what it does know, it must distinguish the relevant from the irrelevant. And what is relevant, it must prioritize in order of importance

Fortunately, at any point in time and space, most of the detail in the environment is irrelevant to the immediate task of the intelligent system. Therefore, the key to building practical intelligent systems lies in understanding how to focus the available computing resources on what is important and ignore what is irrelevant.

3.9.1 Knowing What is Important

The problem of distinguishing what is important from what is irrelevant must be addressed from two perspectives: *top down* and *bottom up*.

Top down: what is important is defined by behavioral goals. The intelligent system is driven by high-level goals and priorities to focus attention on objects specified by the task, using resources identified by task knowledge as necessary for successfully accomplishing given goals. Top down goals and high-level perceptions generate expectations of what objects and events might be encountered during the evolution of the task and which are important to achieving the goal.

Bottom up: what is important is the unexpected, unexplained, unusual, or out-of-limits. At each level, sensory processing functions detect errors between what is expected and what is observed. Error signals are processed at lower levels first. Control laws in lower level behavior generation processes generate corrective actions designed to correct the errors and bring the process back to the plan. However, if low level reactive control laws are incapable of correcting the differences between expectations and observations, errors filter up to higher levels where plans may be revised and goals restructured. The lower levels are thus the first to compute attributes of signals or images that indicate problems or emergency conditions, such as limits being exceeded on position, velocity, acceleration, vibration, pressure, force, current, voltage, or

temperature. The lower levels of control are also the first to act to correct, or compensate for errors.

In either top down or bottom up, hierarchical leveling provides a mechanism for focusing the computational resources of the lower levels on particular regions of time and space. Higher level nodes with broad perspective and long planning horizon determine what is important, while the lower levels detect anomalies and attend to details of correcting errors and following plans. In each node at each level, computing resources are focused on issues relevant to the decisions that must be made within the scope of control and time horizon of that node.

The region in space and time that is most relevant to the behavioral choices of an intelligent system is the region around the "here and now." An intelligent system exists at the center of its own egosphere. The relevance of entities and events in the world are usually inversely proportional to their distance from the origin of this egosphere (i.e., here). The intelligent system also exists at the point in time labeled "now" (i.e., t = 0). The relevance of events is usually inversely proportional to their time from "now."

3.9.2 Focusing on What is Important

Focusing of attention can be accomplished through masking, windowing, and filtering based on object and feature hypotheses and task goals. It can also be accomplished by pointing high resolution regions of sensors at objects-of-attention. For example in the human eye, the visual field is sampled at high resolution in the foveal region, and lower resolution in the periphery. Similarly, tactile sensors are closely spaced to produce high resolution in the fingertips, lips, and tongue with much lower resolution in other regions of the skin. The foveal area of the eyes and the high resolution tactile sensory regions of the fingers and lips are behaviorally positioned so as to apply the maximum number of sensors to objects of attention. High resolution sensors are scanned over the world to explore the regions of highest interest to the goals of the task. The result is to make the largest possible number of high resolution measurements of the most important entities and events in the environment and to ignore or sample at lower resolution those entities and events that are considered unimportant.

Thus, at each level in the 4D/RCS sensory processing hierarchy, attention is used to mask, filter, and window sensory data and to focus computational resources on objects and events that are important to the mission goal. This keeps the computational load of processing sensory data within manageable limits at all levels of the hierarchy.

3.10 A Notional 4D/RCS Concept for FCS

To illustrate the types of issues that can be addressed using the 4D/RCS architecture, an example is given below of an eight-level hierarchy for a FCS battalion based on a merger of two notional concepts. One is the notional FCS Organization Concept developed by the FY2000 Summer Study for the Army Science Board based on a Ft. Knox Mounted Maneuver Battle Lab experimental force design. [Army 00] The second notional concept is the Lead Systems Integrator concept expressed in the Boeing Broad Industry Announcement. [Boeing 02] The

specific numbers and functions described in this example are illustrative only. Exact numbers will be determined by future FCS design studies.

Level 8 – Battalion

A notional FCS battalion might be an organization consisting of a headquarters unit, two fighting vehicle companies, two infantry vehicle companies, a reconnaissance platoon, a net fires platoon, and a support company. A computational node at level 8 of the 4D/RCS architecture corresponds to a battalion headquarters unit housed within two 16 ton (142.3 kN) command, control, and communications (C3) vehicles that enable C3 on the move for the battalion. The battalion C3 vehicles each include a driver, a commander, and a 4-soldier workstation.

Incoming orders to the battalion headquarters are decomposed, by staff or on-board computers according to the FCS configuration at the time, into assignments for each of the subordinate units within the battalion. Resources and assets are allocated to each subordinate unit, and a schedule is generated for each unit to maneuver and carry out assigned operations. Together, these assignments, allocations, and schedules comprise a battalion level plan. The plan may be devised by the battalion commander alone, or in consultation with his company commanders. The battalion level planning process may consider the objectives and constraints of the incoming orders, the best time and place to engage the enemy, the exposure of each unit's movements to enemy observation, and the traversability of roads and cross-country routes. The battalion commander typically defines the rules of engagement for the units under his command and works with his company commanders to develop a schedule that meets the objectives of the mission orders given to the battalion. In the 4D/RCS battalion node, plans are computed for a period of about 24 hours (h) and recomputed at least once every 2 h (or more frequently if necessary).

In the battalion node, the 4D/RCS world modeling process maintains a knowledge database that contains maps that describe the terrain and location of friendly and enemy forces (to the extent that they are known), and roads, bridges, towns, and obstacles such as mountains, rivers, and woods. Overlaid on the maps are icons that represent objects and organizational units in the environment. These icons have links to symbolic data structures that describe attributes of objects such as class, size, composition, strength, state of readiness, movement, and estimated intent. The battalion level knowledge database may be updated from intelligence reports as well as from sensors on organic ground and air platforms. Maps used for planning typically have a range of at least 100 km x 100 km (i.e. larger than the typical area of concern to the battalion) with a resolution of about 30 m, which corresponds to digital terrain elevation data (DTED) level 2.

Sensory processing in the battalion HQ node integrates information about the movement of forces, the level of supplies, and the operational status of all the units in the battalion, plus intelligence about enemy units in the area of concern to the battalion. This information is used to update maps and data in the knowledge database so as to keep it accurate and current. The battalion node also contains value judgment functions that enable the battalion commander to evaluate the cost, risk, and benefit of various tactical options.

Operator interfaces allow human operators and commanders to visualize information such as the deployment and movement of forces, the availability of ammunition, and the overall

situation within the scope of attention of the battalion commander. The commander can intervene at any time to change priorities, alter tactics, or redirect the allocation of resources.

Output from the battalion level consists of commands to the company level. New commands typically consist of tasks expected to require about 5 h to complete. These may be issued at any time. Company commanders attached to the battalion are expected to convey commands to their respective units, monitor how well their company is following the battalion plan, and make adjustments as necessary to keep on plan.

Level 7 – Company

A FCS company is a unit typically consisting of three platoons that may include fighting vehicles, armored personnel carriers, artillery, and logistics. For example, a fighting vehicle company may consist of two fighter platoons, one infantry platoon, and two mortar vehicles. Each infantry company consists of two infantry platoons, and one fighter platoon, and two mortar vehicles. Each support company consists of several resupply vehicles, one or more recovery vehicles to provide towing and recovery assistance, one or more medical vehicles, and one or more mobility/counter-mobility vehicles to breach or lay mine fields.

A computational node at level 7 of the 4D/RCS architecture corresponds to a company headquarters unit housed in two 16 ton C3 vehicles. The company C3 vehicles each consist of a driver, a commander, and a 4-soldier operator interface workstation. Each company headquarters unit plans activities and allocates resources for the units attached to the company. Incoming orders to the company are decomposed by the company headquarters into assignments for the subordinate units that belong to the company. Resources and assets are allocated to each unit, and a schedule is generated for each unit to maneuver and carry out assigned operations. Together, these assignments, allocations, and schedules comprise a company-level plan. The plan may be devised by the company commander alone, or in consultation with his platoon leaders. The company level planning process may consider the objectives of the incoming orders, the best time and place to engage the enemy, the exposure of each unit's movements to enemy observation, and the traversability of roads and cross-country routes. The company commander typically defines the rules of engagement for the units under his command and works with his unit leaders to develop a schedule that meets the objectives of the orders given to the company. In the 4D/RCS company node, plans are computed for a period of about 5 h and recomputed at least once every 30 min (or more frequently if necessary).

In the company node, the 4D/RCS world modeling process maintains a knowledge database that contains maps that describe the terrain and location of friendly and enemy forces (to the extent that they are known), and roads, bridges, towns, and obstacles such as mountains, rivers, and woods. Overlaid on the maps are icons that point to symbolic data structures that describe attributes such as strength, state of readiness, movement, and estimated intent. The level knowledge database may be updated from intelligence reports as well as from sensors on organic ground and air platforms. Maps used for planning typically have a range of 30 km x 30 km (i.e. larger than the typical area of concern to the company) with a resolution of about 30 m.

Sensory processing in the company node integrates information about the movement of forces, the level of supplies, and the operational status of all the units in the company, plus intelligence about enemy units in the area of concern to the company. This information is used to update maps and symbolic data structures in the knowledge database so as to keep it accurate

and current. The company node also contains value judgment functions that enable the company commander to evaluate the cost, risk, and benefit of various tactical options.

An operator interface allows human operators (either on-site or remotely) to visualize information such as the deployment and movement of forces, the availability of ammunition, and the overall situation within the scope of attention of the company commander. The operator can intervene to change priorities, alter tactics, or redirect the allocation of resources.

Output from the company level consists of input commands to the platoon level. New commands typically consist of tasks expected to require about 1 h to complete. These may be issued at any time. Platoon leaders are expected to convey commands to their respective units, monitor how well their platoon is following the company plan, and make adjustments as necessary to keep on plan.

Level 6 – Platoon

A FCS platoon attached to a fighting company or infantry company may consist of a headquarters unit housed in a 16 ton C3 vehicle with a human driver, commander, and a 4 soldier workstation. The remainder of the platoon consists of six or more vehicles of the following type in some combination:

- A 16 ton armored personnel carrier with a human driver and commander for transporting a full 10-man infantry squad and associated equipment
- A 16 ton vehicle with a human driver and commander, armed with a 105 mm to 120 mm cannon for line of sight and beyond line of sight engagement up to 15 km
- A 16 ton vehicle with a human driver and commander, armed with a non-line of sight weapon with 120 mm to 155 mm projectiles with 30 km to 40 km range
- A 16 ton vehicle with a human driver and commander, armed with a 120 mm mortar mounted on a turret for precision-guided mortar munitions
- A 16 ton vehicle with a human driver and commander, armed with a 25 mm to 50 mm chain gun
- A 6 or 16 ton vehicle with a human driver and commander, armed with a non-lethal weapons system
- Several 6 or 16 ton semi-autonomous robot mule vehicles
- A number of soldier robots

A RSTA platoon attached to FCS battalion may consist of a headquarters unit housed in a 16 ton C3 vehicle with a driver and a commander. Also in the command vehicle would be a RSTA suite with a 2 soldier workstation, a 5 meter mast, FLIRs, day/night TV, 10 km laser range finder, Ka band radar, and 360 degree all elevation pan/tilt. The remainder of the RSTA platoon would consist of:

- A 16 ton launcher for small AUVs with a pod of 32 small AUVs and launching system
- A 16 ton UGV/UAV control vehicle with a 4 soldier workstation for control of UGVs and UAVs
- One or more 6 ton armed reconnaissance vehicles with 2 meter masts, RSTA suites, and Hell Fire missiles or 35 mm chain guns
- Several 1 ton scout vehicles with RSTA suites

- One or more networks of unattended ground sensors
- Several soldier robots

A Net Fires platoon attached to a FCS battalion may consist of a headquarters unit housed in a 16 ton C3 vehicle with a driver and commander. Also in the C3 vehicle would be a 4 soldier workstation. The remainder of the Net Fires platoon would consist of four 6 or 16 ton missile launchers with BLOS precision guided missiles and loitering munitions with 40 km to 150 km range.

A Support platoon attached to a FCS support division may consist of a headquarters unit housed in a 16 ton C3 vehicle with a human driver, a commander, and a 4 soldier workstation. The remainder of the platoon consists of:

- One or more resupply vehicles that can be configured as a semi-autonomous leader-follower vehicles
- One or more recovery vehicles that provides towing and recovery assistance
- One or more semi-autonomous mobility/counter-mobility vehicles to breach and lay minefields
- Several semi-autonomous mule vehicles
- Several medical vehicles with station for a medical corpsman
- Zero or more bridging vehicles equipped to lay bridges

A 4D/RCS node at the Platoon level corresponds to a platoon headquarters unit. The platoon commander and section leaders plan activities and allocate resources for the sections in the platoon. Platoon orders are decomposed into job assignments for each section. Resources are allocated, and a schedule of activities is generated for each section. Tactical maneuvers are planned relative to major terrain features and other sections within the platoon. Inter-section formations are selected on the basis of tactical goals, stealth requirements, and other priorities. At the platoon level, plans are computed for a period of about 1 h into the future, and replanning is done about every 5 min, or more often if necessary. Section waypoints about 10 min apart are computed.

The surrogate platoon node in each vehicle performs the functions of the platoon headquarters unit when the vehicle is not in direct communication with the chain of command. It plans activities for the vehicle on a platoon level time scale and estimates what vehicle level maneuvers should be executed in order to follow that plan. Movements are planned relative to major terrain features and other vehicles within the platoon.

At the platoon level, the 4D/RCS world model contains maps with a range of about 10 km and resolution of about 30 m that describe the location of objectives and the routing between them. These maps are overlaid with icons with pointers to a symbolic database that contains names and attributes of targets, and the weapons and ammunition necessary to attack them.

Sensory processing integrates intelligence about the location and status of friendly and enemy forces. Value judgment evaluates tactical options for achieving section objectives. An operator interface allows human operators to visualize the status of operations and the movement

of vehicles within the section formation. Operators can intervene to change priorities and reorder the plan of operations.

Output from the platoon level consists of input commands to the section level. New commands typically consist of tasks expected to require about 10 min to complete. These may be issued at any time. Section commanders (i.e., platoon level executors) are expected to convey commands to their respective units, monitor how well their section is following the platoon plan, and make adjustments as necessary to keep on plan.

Level 5 – Section

A section is a unit that consists of a group of individual scout vehicles such as HMMWVs and UGVs. A 4D/RCS node at the section level corresponds to a section leader and vehicle commanders within the section. The section leader assigns duties to the vehicles in his section and coordinates the scheduling of cooperative activities between vehicles within a section. Orders are decomposed into assignments for each vehicle, and a schedule is developed for each vehicle to maneuver within assigned corridors taking advantage of local terrain features and avoiding obstacles. Plans are developed to conduct coordinated maneuvers and to perform reconnaissance, surveillance, or target acquisition functions. At the section level, plans are computed for about 10 min into the future, and replanning is done about every 1 min, or more often if necessary. Vehicle waypoints about 1 min apart are computed.

At the section level, the 4D/RCS world model symbolic database contains names, coordinates, and other attributes of other vehicles within the section, other sections, and potential enemy targets. Maps with a range of about 2 km and a resolution of about 30 m are typical. Maps at the section level describe the location of vehicles, targets, landmarks, and local terrain features such as buildings, roads, woods, fields, streams, fences, ponds, etc. Sensory processing determines the position of landmarks and terrain features, and tracks the motion of groups of vehicles and targets. Value judgment evaluates plans and computes cost, risk, and payoff of various alternatives. An operator interface allows human operators to visualize the status of the battlefield within the scope of the section, or to intervene to change priorities and reorder the sequence of operations or selection of targets. Vehicle commanders issue commands to their respective vehicles, monitor how well plans are being followed, and make adjustments as necessary to keep on plan. Output commands to individual vehicles to engage targets or maneuver relative to landmarks or other vehicles may be issued at any time, but on average are planned for tasks that last about 1 min.

Surrogate section, platoon, and battalion nodes in each UGV perform the functions of higher level command echelons when the UGV is not in direct communication with its chain of command. Each surrogate node plans activities for the UGV on a time scale commensurate with planning activities in the respective higher level echelons, and estimates what vehicle level maneuvers should be executed in order to follow those plans.

Level 4 – Individual Vehicle

The vehicle is a unit that consists of a group of subsystems, such as mobility, attention, communication, and mission package. A manned scout vehicle may have a driver, vehicle commander, and a lookout. Thus, a 4D/RCS node at the vehicle level corresponds to a vehicle

commander plus subsystem planners and executors. The vehicle commander assigns jobs to subsystems and schedules the activities of all the subsystems within the vehicle. A schedule of waypoints is developed by the mobility subsystem to avoid obstacles, maintain position relative to nearby vehicles, and achieve desired vehicle heading and speed along the desired path on roads or cross-country. A schedule of tracking activities is generated for the attention subsystem to track obstacles, other vehicles, and targets. A schedule of activities is generated for the mission package and the communication subsystems. Waypoints and task activities about 5 s apart out to a planning horizon of 1 min are replanned every 5 s, or more often if necessary.

At the vehicle level, the world model symbolic database contains names (identifiers) and attributes of objects, such as: the size, shape, and surface characteristics of roads, ground cover, or objects such as rocks, trees, bushes, mud, and water. Maps are generated from on-board sensors with a range of about 500 m and resolution of 4 meters. These maps are registered and overlaid with 30 meter resolution map data from Section level maps. Maps represent object positions (relative to the vehicle) and dimensions of road surfaces, buildings, trees, craters, and ditches. Sensory processing measures object dimensions and distances, and computes relative motion. Value judgment evaluates trajectory planning and sensor dwell time sequences. An operator interface allows a human operator to visualize the status of operations of the vehicle, and to intervene to change priorities or steer the vehicle through difficult situations. Subsystem controller executors sequence commands to subsystems, monitor how well plans are being followed and modify parameters as necessary to keep on plan. Output commands to subsystems may be issued at any time, but typically are planned to change only about once every 5 s.

Level 3 – Subsystem Level

Each subsystem node is a unit consisting of a controller for a group of related Primitive level sub-subsystems. A 4D/RCS node at the Subsystem Level assigns jobs to each of its Primitive sub-subsystems and coordinates the activities among them. A schedule of Primitive mobility waypoints and Primitive mobility actions is developed to avoid obstacles. A schedule of pointing commands is generated for aiming cameras and sensors. A schedule of messages is generated for communications, and a schedule of actions is developed for operating the mission package sub-subsystems. The Primitive mobility way points are about 500 ms apart out to a planning horizon of about 5 s in the future. A new plan is generated about every 500 ms.

At the Subsystem level, the world model symbolic database contains names and attributes of environmental features such as: road edges, holes, obstacles, ditches, and targets. Vehicle centered maps with a range of 50 meters and resolution of 40 cm are generated using data from range sensors. These maps represent the shape and location of terrain features and obstacle boundaries. Sensory processing computes surface properties such as dimensions, area, orientation, texture, and motion. Value judgment supports planning of steering and aiming computations, and evaluates sensor data quality. An operator interface allows a human operator to visualize the state of the vehicle, or to intervene to change mode or interrupt the sequence of operations. Subsystem executors compute at a 5 Hz clock rate. They sequence commands to primitive systems, monitor how well plans are being followed, and modify parameters as necessary to keep on plan. Output commands to Primitive sub-subsystems may be issued at any 200 ms interval, but typically are planned to change on average about once every 500 ms.

Level 2 – Primitive Level

Each node at the Primitive level is a unit consisting of a group of controllers that plan and execute velocities and accelerations to optimize dynamic performance of components such as steering, braking, acceleration, gear shift, camera pointing, and weapon loading and pointing, while taking into consideration dynamical interaction between mass, stiffness, force, and time. Communication messages are encoded into words and strings of symbols. Velocity and acceleration set points are planned every 50 ms out to a planning horizon of 500 ms.

The world model symbolic database contains names and attributes of state variables and features such as target trajectories and edges of objects. Maps are generated from camera data. Five-meter maps have a resolution of about 4 cm. Driving plans can be represented by predicted tire tracks on the map, and visual attention plans by predicted fixation points in the visual field.

Sensory processing computes linear image features such as occluding edges, boundaries, and vertices and detects strings of events. Value judgment cost functions support dynamic trajectory optimization. An operator interface allows a human operator to visualize the state of each controller, and to intervene to change mode, to override velocities, or to teleoperate the vehicle. Primitive level executors keep track of how well plans are being followed, and modify parameters as necessary to keep within tolerance. Primitive executors compute at a 20 Hz clock rate. Output commands are issued to the Servo level to adjust set points for vehicle steering, velocity, and acceleration or for pointing sensors or weapons platforms. Output commands are issued every 50 ms.

Level 1 – Servo Level

Each node at the servo level is a unit consisting of a group of controllers that plan and execute actuator motions and forces, and generate discrete outputs. Communication message bit streams are produced. The servo level transforms commands from component to actuator coordinates and computes motion or torque commands for each actuator. Desired forces, velocities, and discrete outputs are planned for 5 ms intervals out to a planning horizon of 50 ms.

The world model symbolic database contains values of state variables such as actuator positions, velocities, and forces, pressure sensor readings, position of switches, and gear shift settings. Sensory processing detects events and scales and filters data from individual sensors that measure position, velocity, force, torque, and pressure. Sensory processing also computes pixel attributes in images such as spatial and temporal gradients, stereo disparity, range, color, and image flow. An operator interface allows a human operator to visualize the state of the machine, or to intervene to change mode, set switches, or jog individual actuators. Executors cause servo actuators and motors to follow planned trajectories. Position, velocity, or force servoing may be implemented, and in various combinations. Servo executors compute at a 200 Hz clock rate. Motion output commands to power amplifiers specify desired actuator torque or power every 5 ms. Discrete output commands produce switch closures and activate relays and solenoids.

The above example illustrates how the 4D/RCS multilevel hierarchical architecture assigns different responsibilities and duties to various levels of the hierarchy with different range and resolution in time and space at each level. At each level, sensory data is processed, entities

are recognized, world model representations are maintained, and tasks are decomposed into parallel and sequential subtasks, to be performed by cooperating sets of agents. At each level, feedback from sensors reactively closes a control loop allowing each unit at each level to respond and react to unexpected events.

At each level, there is a characteristic range and resolution in space and time, a characteristic bandwidth and response time, and a characteristic planning horizon and level of detail in plans. *The 4D/RCS architecture thus organizes the planning of behavior, the control of action, and the focusing of computational resources such that RCS_NODEs at each level have a limited amount of responsibility and a manageable level of complexity.*

3.11 4D/RCS for Demo III

There are three ways to visualize a 4D/RCS hierarchy. These are illustrated in Figure 6.

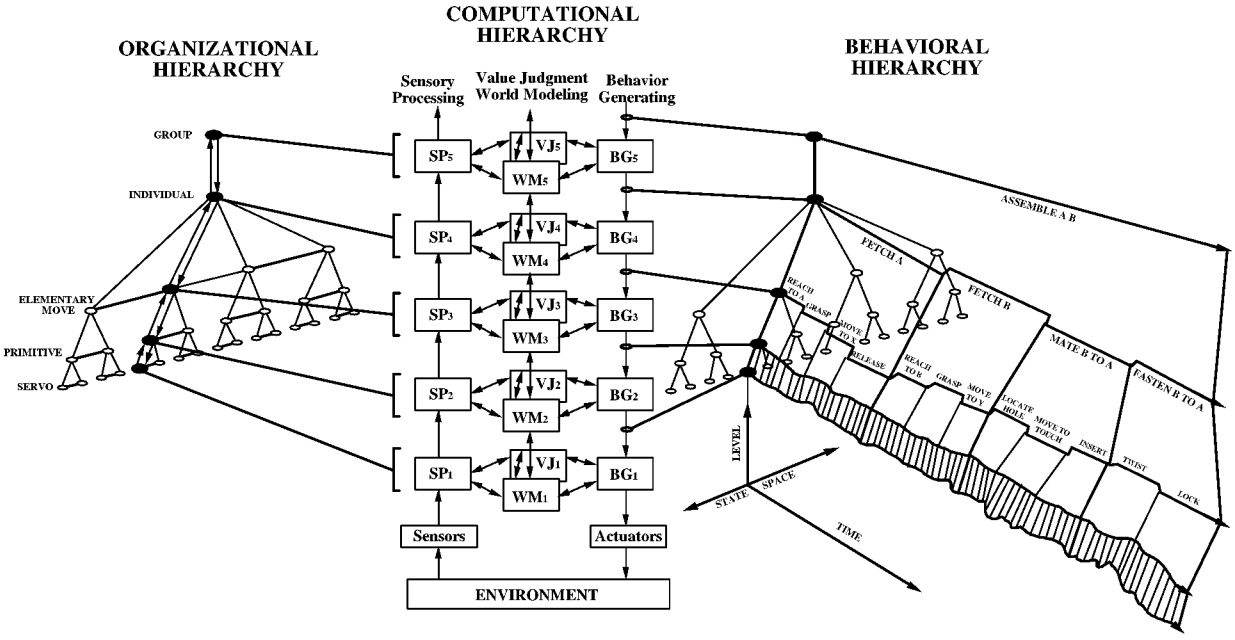


Figure 6. Three views of the 4D/RCS architecture. The organizational hierarchy shows the RCS_NODES arranged as a hierarchy of organizational units. The computational hierarchy shows the internal structure of the nodes in single chain of command. The behavioral hierarchy shows the time history of commands that flow in a chain of command over a period of time.

Figure 7 is a computational hierarchy view of the first five levels in the chain of command containing the Autonomous Mobility Subsystem in the 4D/RCS architecture developed for Demo III. On the right of Figure 7, Behavior Generation (consisting of Planner and Executor) decompose high level mission commands into low level actions. The text inside the Planner at each level indicates the planning horizon at that level.

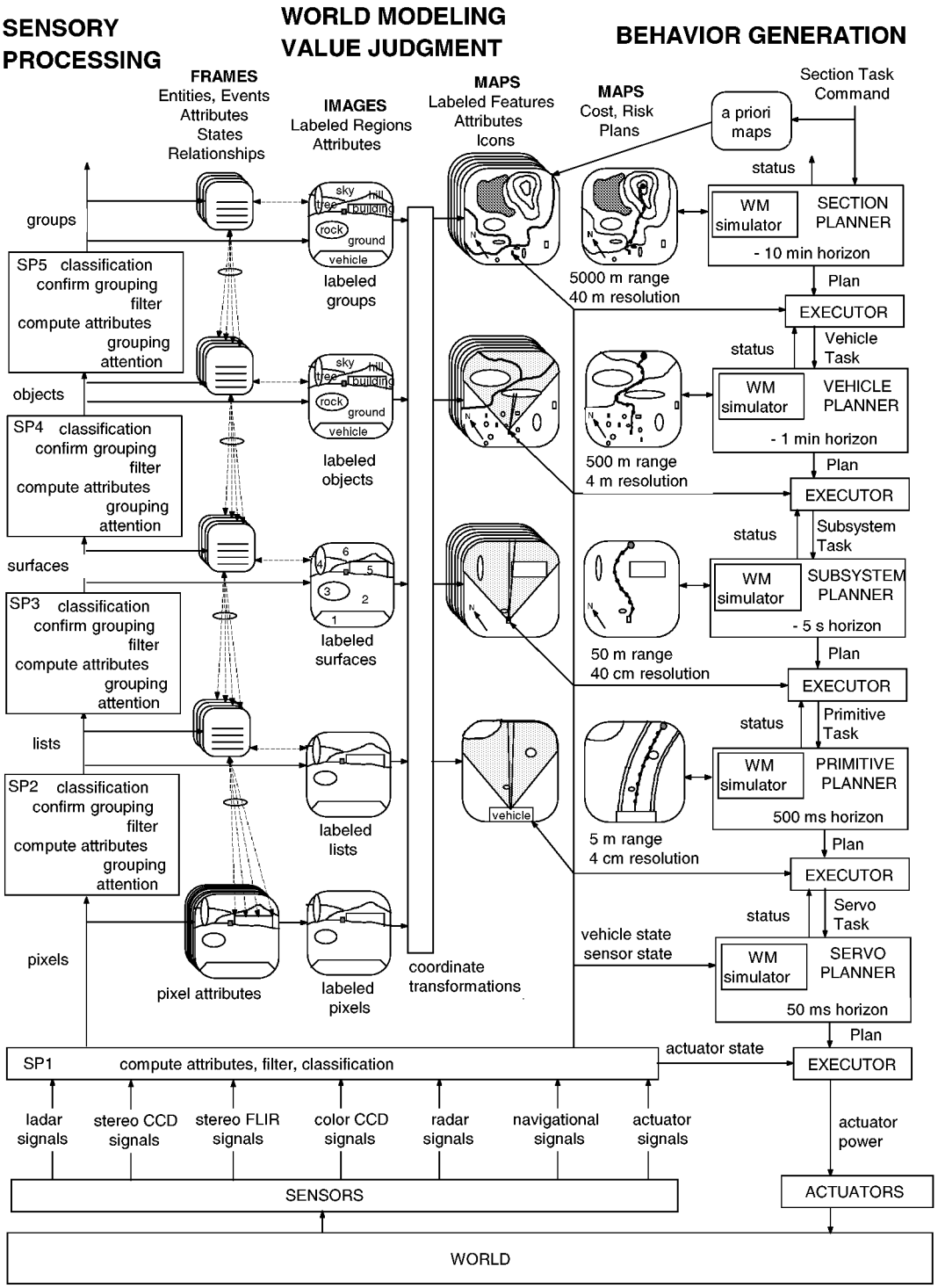


Figure 7. Five levels of the 4D/RCS architecture for Demo III. On the right are Planner and Executor modules. In the center are maps for representing terrain features, road, bridges, vehicles, friendly/enemy positions, and the cost and risk of traversing various regions. On the left are Sensory Processing functions, symbolic representations of entities and events, and segmented images with labeled regions. The coordinate transforms in the middle use range information to assign labeled regions in the entity image hierarchy on the left to locations on planning maps on the right. This causes the entity class hierarchy on the left to be orthogonal to the BG process hierarchy on the right.

The Executor at each level executes the plan generated by the Planner. Meanwhile, the Planner is replanning based on an updated world state. Each planner has a world model simulator that is appropriate for the problems encountered within the node at its level. The Planners and Executors operate asynchronously. At each level, the Planner generates a new plan and the Executor outputs new commands to subordinates on the order of ten times within each planning horizon. At each lower level, the planning horizons shrink by a factor of about ten. The relative timing relationships between levels are designed to facilitate control stability and smooth transitions among hierarchical control levels. The timing numbers in Figure 7 are illustrative only. The actual rates may differ in different applications.

In the center of Figure 7, each map has a range and resolution that is appropriate for path planning at its level. At each level, there are symbolic data structures and segmented images with labeled regions that describe entities, events, and situations that are relevant to decisions that must be made at that level. On the left is a sensory processing hierarchy that extracts information from the sensory data stream that is needed to keep the world model knowledge database current and accurate.

At the bottom of Figure 7 are actuators that act on the world and sensors that measure phenomena in the world. The Demo III XUVs are designed to accommodate a variety of sensors that include a LADAR, stereo CCD cameras, stereo FLIRs, a color CCD, vegetation penetrating radar, GPS (Global Positioning System), an inertial navigation package, actuator feedback sensors, and a variety of internal sensors for measuring parameters such as engine temperature, speed, vibration, oil pressure, and fuel level. The XUVs also may carry a Reconnaissance, Surveillance, and Target Acquisition (RSTA) package that includes long-range cameras and FLIRs, a laser range finder, and an acoustic package.

In Figure 7, the bottom (Servo) level has no map representation. The Servo level deals with actuator dynamics and reacts to sensory feedback from actuator sensors. The Primitive level map has range of 5 m with resolution of 4 cm. This enables the vehicle to make small path corrections to avoid bumps and ruts during the 500 ms planning horizon of the Primitive level. The Primitive level also uses accelerometer data to control vehicle dynamics and prevent roll-over during high speed driving. The Subsystem level map has range of 50 m with resolution of 40 cm. This map is used to plan about 5 s into the future to find a path that avoids obstacles and provides a smooth and efficient ride. The Vehicle level map has a range of 500 m with resolution of 4 m. This map is used to plan paths about 1 min into the future taking into account terrain features such as roads, bushes, gullies, or tree lines. The Section level map has a range of 2 km with resolution of about 30 m. This map is used to plan about 10 min into the future to accomplish tactical behaviors. Higher level maps (not shown in Figure 7) can be used to plan platoon, company, and battalion missions lasting about 1 h, 5 h, and 24 h respectively. These are derived from military maps and intelligence provided by the digital battlefield database.

At all levels, 4D/RCS planners are designed to generate new plans well before current plans become obsolete. Thus, action always takes place in the context of a recent plan, and feedback through the executors closes reactive control loops using recently selected control parameters. To meet the demands of dynamic battlefield environments, the 4D/RCS architecture specifies that replanning should occur within about one-tenth of the planning horizon at each level.

Executors are designed to react to sensory feedback even faster than the replanning interval. The Executors monitor feedback from the lower levels on every control cycle. Whenever an Executor senses an error between its output CommandGoal and the predicted state (status from the subordinate BG Planner) at the GoalTime, it may react by modifying the commanded action so as to cope with that error. This closes a feedback loop through the Executor at that level within a specified reaction latency.

3.12 The Inter-Node Interactions within a Hierarchy

Sensory processing and behavior generation are both hierarchical processes, and both are embedded in the nodes that form the 4D/RCS organizational hierarchy. However, the SP and BG hierarchies are quite different in nature and are not directly coupled. Behavior generation is a hierarchy based on the decomposition of tasks and the assignment of tasks to operational units. Sensory processing is a hierarchy based on the grouping of signals and pixels into entities and events. In 4D/RCS, the hierarchies of sensory processing and behavior generation are separated by a hierarchy of world modeling processes. The WM hierarchy provides a buffer between the SP and BG hierarchies with interfaces to both.

The WM interface with the SP hierarchy is designed to compare observations with predictions. This requires that WM predictions be at the same level of abstraction and in the same coordinate frame as SP observations at each level. On the other side, the WM interface with the BG hierarchy is designed to support task decomposition and planning. This requires that WM representations be at the same level of range and resolution in space and time, and be in the same coordinate system as the tasks being decomposed at each level. Figure 7 illustrates how the world modeling processes can be designed to fulfill both these requirements.

Note that in Figure 7, the left side of the world modeling hierarchy maintains a hierarchy of entity images that are linked to a hierarchy of symbolic frames. These represent a hierarchy of entities with increasing degree of aggregation (i.e., pixels, list entities, surface entities, object entities, etc.) Yet the right side of the world modeling hierarchy maintains a hierarchy of maps with increasing range and decreasing resolution. It is in the middle of the world modeling hierarchy that a coordinate transformation process converts from image coordinates to map coordinates. As a result, entities at any level in the image domain may transform into maps at any level in the planning domain. For example, an entity near the bottom of an image (short range) might be transformed into a Primitive level map, whereas another pixel in the same image near the horizon (long range) might transform into a Vehicle or Section level map. Thus, where an entity in the image ends up in the map depends not on its level in the SP hierarchy of entities, but on its distance from the camera. For example, a pixel or list entity in the image may end up in a Vehicle or Section level map, whereas an object or group entity in the image may end up in a Primitive or Subsystem level map. This flow of information between levels in the WM is represented in the 4D/RCS diagram of Figure 4 by the double-headed arrow marked "To Higher and Lower Level World Modeling" and in Figure 5 by the vertical pathways between WM processes.

3.13 Command Vocabularies

A command vocabulary is the set of named actions or tasks that a 4D/RCS behavior generation (BG) process can perform. Each BG process at each level of the control hierarchy has its own unique command vocabulary. Examples of the command vocabularies at various levels of the Demo III hierarchy are:

Section Level Commands

- ☐ Init
- ☐ E-stop
- ☐ Pause/Resume (task T)
- ☐ Abort
- ☐ RetroTraverse (to point P)
- ☐ CooperativeSearch (of area A)
- ☐ PerformRouteReconnaissance (along route R)
- ☐ PerformAreaReconnaissance (of area A)
- ☐ ConductScreen(for unit U)
- ☐ PerformObstacleRestrictedRecon(over area A)
- ☐ ReconnoiterBuiltUpArea(area A)
- ☐ ConductTacticalMovement(to point P)
- ☐ ConductTacticalRoadMarch(along route R)
- ☐ EstablishObservationPost(at point P)

Section level commands are expressed in UTM WGS84 world coordinates. Parameters may include goal positions to be occupied, desired paths to be traversed, required regions to be observed. Parameters may also include specifications for performance such as speed, time of completion, required precision, and choice of formation (e.g., line, wedge, vee, and column, staggered). Mode parameters may include level of aggressiveness, priority, probability of enemy contact, and acceptable risk or cost. Constraint parameters may specify corridor boundaries and speed limit. Condition parameters may specify what is required to begin or continue. Typical intervals between Section level commands are 10 minutes.

Vehicle Level Commands

- ☐ Init
- ☐ E-stop
- ☐ Pause/Resume (task T)
- ☐ Abort
- ☐ RetroTraverse(to x, y by t)
- ☐ SendImage(between x1, y1 and x2, y2)
- ☐ ReportStatus
- ☐ NavigateToGoalPoint(at x, y by t)
- ☐ PerformRoadMarch(to x, y by t)
- ☐ OccupyOverwatchPosition(at x, y by t)
- ☐ OccupyObservation/ListeningPost(at x, y by t)

- ☐ DetectBarriersToMovement(between x1, y1 and x2, y2)
- ☐ ReconnoiterArea(between x1, y1 and x2, y2)
- ☐ ReconnoiterRoute(from x1, y1 to x2, y2 by t)
- ☐ LocateBypassOfArea(between x1, y1 and x2, y2)
- ☐ ReconnoiterTerrain(between x1, y1 and x2, y2)
- ☐ ReconnoiterDefilesOnRoute(from x1, y1 to x2, y2 by t)
- ☐ ReconnoiterLateralRoutesAlongRoute(from x1, y1 to x2, y2 by t)
- ☐ ReconnoiterApproachToRoute(from x1, y1 to x2, y2 by t)
- ☐ IdentifyVehicles&Personnel(between az1 and az2)
- ☐ IdentifyThreatVehicles(between az1 and az2)
- ☐ MoveToMaintainContact(with target)
- ☐ Hide&MaintainContact(with target)
- ☐ HideFromEnemy(between bearing1 and bearing2)

Vehicle level commands are expressed in vehicle-centered, North-oriented, world coordinates. Parameters typically specify where, when, how fast, and how important the task is. Typical interval between Vehicle level commands is 50 s.

Autonomous Mobility Subsystem Level Commands

- ☐ Init
- ☐ E-stop
- ☐ Pause/Resume
- ☐ Abort
- ☐ RetroTraverse(to position, velocity, heading by t)
- ☐ TurnAround(position, velocity, heading by t)
- ☐ BackUp(to position, velocity, heading by t)
- ☐ GoWithinCorridorTo(position, velocity, heading, right boundary, left boundary by t)
- ☐ GoToRoad(position at velocity or by t)
- ☐ GoOnRoadTo(position at velocity in lane by t)
- ☐ GoBesideRoadTo(position at velocity, offset until t)
- ☐ GoStealthyTo(position, velocity, heading by t)
- ☐ GoToHillCrest(position, heading by t)
- ☐ LeaveHillCrest(position, heading by t)
- ☐ GoToFeature(feature, position, heading by t)
- ☐ DashTo(position, velocity, heading at t)
- ☐ Hide(position, heading by t)
- ☐ HullDown(position, heading)
- ☐ StopAt(phase line by t)
- ☐ ScanTreeLine(bearing, elevation, length)
- ☐ ConductSecurityHalt(position, heading at t)

Subsystem level commands are expressed in vehicle-centered, vehicle-oriented, world coordinates. Parameters may specify position, velocity, heading, and timing requirements. Typical interval between Subsystem level commands is 5 s.

Primitive Level – Primitive Driver Commands

- ☐ Init
- ☐ E-stop
- ☐ Pause/Resume
- ☐ Abort
- ☐ GoTo(position, velocity, heading by t)
- ☐ FollowLeadVehicle(at distance)

Primitive Level – Gaze Commands

- ☐ Init
- ☐ E-stop
- ☐ Pause/Resume
- ☐ Abort
- ☐ FixatePoint(at range, bearing, elevation)
- ☐ TrackObject(at range, bearing, velocity at t)
- ☐ ScanTrajectory(from range1, bearing1, elevation1 to range2, bearing2, elevation2)

Primitive level commands are expressed in vehicle-centered, vehicle-oriented, coordinates. Typical interval between Primitive level commands is 500 ms.

Servo Level – Drive Commands

- ☐ Init
- ☐ E-stop
- ☐ Pause/Resume
- ☐ Abort
- ☐ GoTo(range, bearing, speed, heading by t)

Servo Level – Look Commands

- ☐ Init
- ☐ E-stop
- ☐ Pause/Resume
- ☐ Abort
- ☐ GoTo(range, bearing, speed by t)

Servo level commands are expressed in vehicle-centered, vehicle-oriented, coordinates. The interval between Servo commands is 50 ms.

Actuator Commands

- ☐ Init
- ☐ E-stop
- ☐ Abort

- GoTo(position at t)
- GoAt(velocity at t)
- ExertForce(amount at t)

Actuator commands are expressed in actuator coordinates. The interval between actuator commands is 5 ms.

3.14 Command and Plan Structure

In each BG module, commands are decomposed into approximately a ten step plan for each of its subordinate BG modules. For each plan, an Executor cycles through the plan issuing commands to the appropriate subordinate BG modules. Commands into each BG module consist of at least six elements:

(1) **ActionCommand (ac1)** describes the action to be performed and may include a set of modifiers such as priorities, mode, path constraints, acceptable cost, and required conditions.

(2) **GoalCommand (gc1)** describes the desired state (or goal state) to be achieved by the action. Mobility system's state typically includes the position, heading, velocity, and turning rate of the system being controlled. The goal may include the name of a target or object that is to be acted upon. It also may include a set of modifiers such as tolerance.

(3) **GoalTime (gt1)** defines the timing constraint on achieving the goal plus modifiers such as tolerance.

(4) **NextActionCommand (ac2)** describes the planned next action to be performed plus modifiers.

(5) **NextGoalCommand (gc2)** describes the planned next goal state to be achieved plus modifiers.

(6) **NextGoalTime (gt2)** describes the timing constraint on achieving the next goal plus modifiers.

If we designate levels in the hierarchy by a superscript and a node index within each level by a subscript, then input to each behavior generation (BG) process is a command data structure of the form:

- $ac1^j_i$ = ActionCommand plus modifiers for BG module i at level j
- $gc1^j_i$ = GoalCommand state plus modifiers for BG module i at level j
- $gt1^j_i$ = GoalTime plus modifiers for when $gc1^j_i$ should be achieved
- $ac2^j_i$ = NextActionCommanded plus modifiers for BG module i at level j
- $gc2^j_i$ = NextGoalCommand state plus modifiers for BG module i at level j
- $gt2^j_i$ = NextGoalTime plus modifiers for when $gc2^j_i$ should be achieved

Figure 8 shows the command and plan structure for the first five levels of a Demo III XUV. Note that plans exist concurrently at all levels, and the data structures containing the plans form buffers between the planners and executors. This allows planners and executors to run asynchronously, and planners can be constantly replanning at all levels simultaneously and independently from execution.

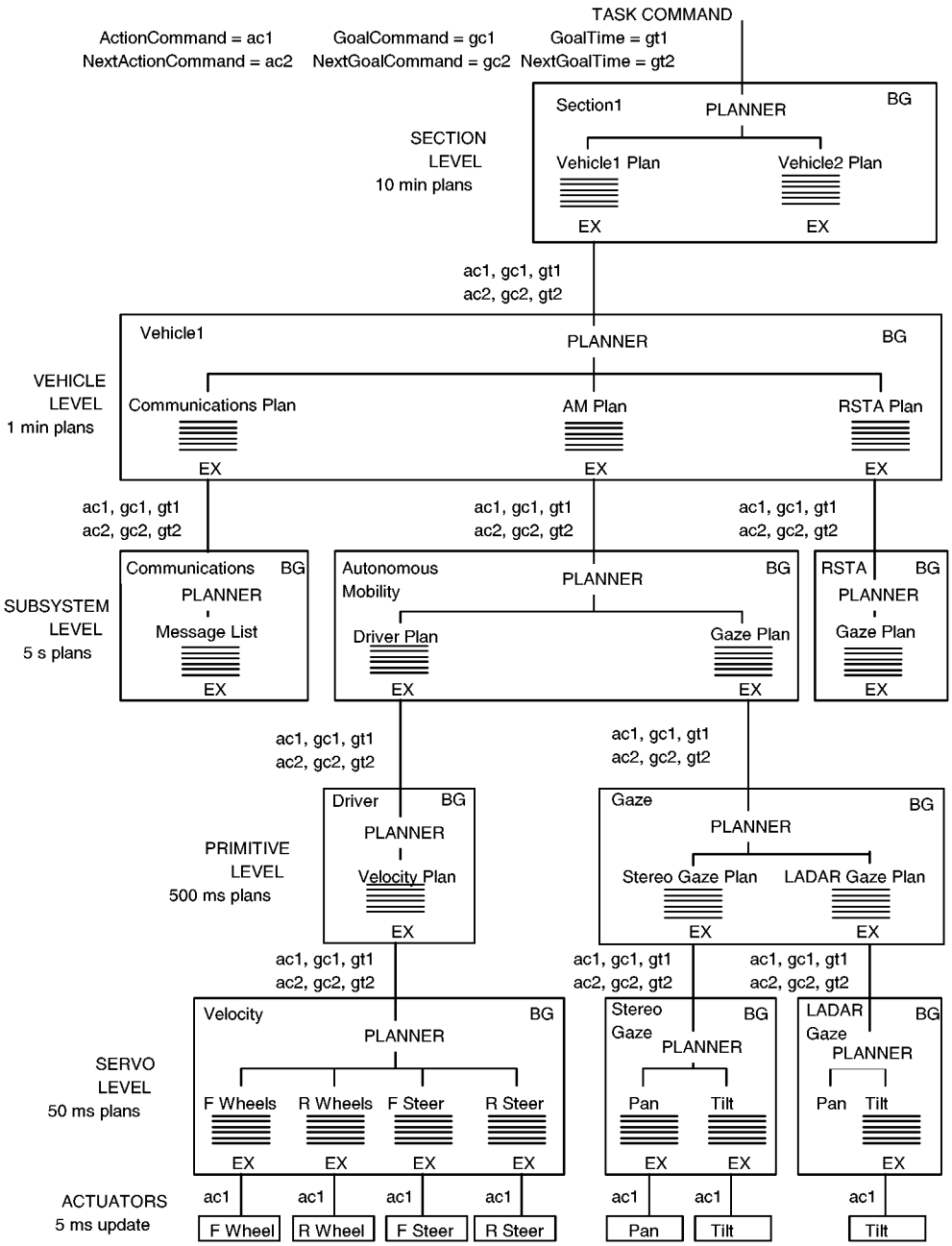


Figure 8. The command and plan structure for Demo III. Note that the plan for each BG module is generated by, and resides in, the BG module above it. For example, the AM Plan for the Autonomous Mobility BG is generated by the Vehicle level planner. The AM plan resides in the Vehicle level BG module and is transformed into commands for the Autonomous Mobility BG by the Vehicle level AM Executor.

Section (level 5)

Commands to Section level BG processes have the form:

Section1 Command Structure

ActionCommand = $ac1^5_1$	GoalCommand = $gc1^5_1$	GoalTime = $gt1^5_1 \approx t + 10$ min
NextActionCommand = $ac2^5_1$	NextGoalCommand = $gc2^5_1$	NextGoalTime = $gt2^5_1 \approx t + 20$ min

where ≈ means equals approximately

The planner in each section level BG process decomposes commands into plans for each of its vehicle BG processes. Each subordinate plan is designed to have about ten steps. For example, a section with two vehicles would have a plan for two vehicles of the form:

Vehicle1 Plan	**Vehicle2 Plan**	**Typical Goal Times**
$ap1^4_1, gp1^4_1, gt1^4_1$	$ap1^4_2, gp1^4_2, gt1^4_2$	$gt1^4_i \approx$ t+1 min
$ap2^4_1, gp2^4_1, gt2^4_1$	$ap2^4_2, gp2^4_2, gt2^4_2$	$gt2^4_i \approx$ t+2 min
$ap3^4_1, gp3^4_1, gt3^4_1$	$ap3^4_2, gp3^4_2, gt3^4_2$	$gt3^4_i \approx$ t+3 min
$ap4^4_1, gp4^4_1, gt4^4_1$	$ap4^4_2, gp4^4_2, gt4^4_2$	$gt4^4_i \approx$ t+4 min
$ap5^4_1, gp5^4_1, gt5^4_1$	$ap5^4_2, gp5^4_2, gt5^4_2$	$gt5^4_i \approx$ t+5 min
$ap6^4_1, gp6^4_1, gt6^4_1$	$ap6^4_2, gp6^4_2, gt6^4_2$	$gt6^4_i \approx$ t+6 min
$ap7^4_1, gp7^4_1, gt7^4_1$	$ap7^4_2, gp7^4_2, gt7^4_2$	$gt7^4_i \approx$ t+7 min
$ap8^4_1, gp8^4_1, gt8^4_1$	$ap8^4_2, gp8^4_2, gt8^4_2$	$gt8^4_i \approx$ t+8 min
$ap9^4_1, gp9^4_1, gt9^4_1$	$ap9^4_2, gp9^4_2, gt9^4_2$	$gt9^4_i \approx$ t+9 min
$ap10^4_1, gp10^4_1, gt10^4_1$	$ap10^4_2, gp10^4_2, gt10^4_2$	$gt10^4_i \approx$ t+10 min

Where:
apk^j_i = action planned for BG module i at level j for plan step k
gpk^j_i = goal planned for BG module i at level j for plan step k
gtk^j_i = goal time planned for BG module i at level j for plan step k
t = time at which the command is scheduled to begin

The GoalTimes shown here illustrate only an approximation, an order of magnitude. Plan steps need not be equally spaced in time or space. There also might be more or fewer than ten steps in a plan.

Vehicle (level 4)

Commands to Vehicle level BG processes would have the form:

Vehicle1 Command Structure

ActionCommand = $ac1^4_1$	GoalCommand = $gc1^4_1$	GoalTime = $gt1^4_1 \approx t + 1$ min
NextActionCommand = $ac2^4_1$	NextGoalCommand = $gc2^4_1$	NextGoalTime = $gt2^4_1 \approx t + 2$ min

A vehicle **with three subsystems** would have a plan for each subsystem of the form:

Autonomous Mobility Plan	**RSTA Plan**	**Communications Plan**	**Goal Times**
$ap1^3_1, gp1^3_1, gt1^3_1$	$ap1^3_2, gp1^3_2, gt1^3_2$	$ap1^3_3, gp1^3_3, gt1^3_3$	$gt1^3_i \approx$ t+5 sec
$ap2^3_1, gp2^3_1, gt2^3_1$	$ap2^3_2, gp2^3_2, gt2^3_2$	$ap2^3_3, gp2^3_3, gt2^3_3$	$gt2^3_i \approx$ t+10 sec

$ap3^3_1$, $gp3^3_1$, $gt3^3_1$	$ap3^3_2$, $gp3^3_2$, $gt3^3_2$	$ap3^3_3$, $gp3^3_3$, $gt3^3_3$	$gt3^3_i \approx$ t+15 sec
$ap4^3_1$, $gp4^3_1$, $gt4^3_1$	$ap4^3_2$, $gp4^3_2$, $gt4^3_2$	$ap4^3_3$, $gp4^3_3$, $gt4^3_3$	$gt4^3_i \approx$ t+20 sec
$ap5^3_1$, $gp5^3_1$, $gt5^3_1$	$ap5^3_2$, $gp5^3_2$, $gt5^3_2$	$ap5^3_3$, $gp5^3_3$, $gt5^3_3$	$gt5^3_i \approx$ t+25 sec
$ap6^3_1$, $gp6^3_1$, $gt6^3_1$	$ap6^3_2$, $gp6^3_2$, $gt6^3_2$	$ap6^3_3$, $gp6^3_3$, $gt6^3_3$	$gt6^3_i \approx$ t+30 sec
$ap7^3_1$, $gp7^3_1$, $gt7^3_1$	$ap7^3_2$, $gp7^3_2$, $gt7^3_2$	$ap7^3_3$, $gp7^3_3$, $gt7^3_3$	$gt7^3_i \approx$ t+35 sec
$ap8^3_1$, $gp8^3_1$, $gt8^3_1$	$ap8^3_2$, $gp8^3_2$, $gt8^3_2$	$ap8^3_3$, $gp8^3_3$, $gt8^3_3$	$gt8^3_i \approx$ t+40 sec
$ap9^3_1$, $gp9^3_1$, $gt9^3_1$	$ap9^3_2$, $gp9^3_2$, $gt9^3_2$	$ap9^3_3$, $gp9^3_3$, $gt9^3_3$	$gt9^3_i \approx$ t+50 sec
$ap10^3_1$, $gp10^3_1$, $gt10^3_1$	$ap10^3_2$, $gp10^3_2$, $gt10^3_2$	$ap10^3_3$, $gp10^3_3$, $gt10^3_3$	$gt10^3_i \approx$ t+60 sec

Subsystem (level 3)

Commands to Subsystem level BG processes would have the form:

Autonomous Mobility Command Structure

ActionCommand = $ac1^3_1$	GoalCommand = $gc1^3_1$	GoalTime = $gt1^3_1 \approx$ t + 5 sec
NextActionCommand = $ac2^3_1$	NextGoalCommand = $gc2^3_1$	NextGoalTime = $gt2^3_1 \approx$ t + 10 sec

A mobility subsystem with Primitive level Driver and Gaze unit controllers would have the form:

Driver Plan	**Gaze Plan**	**Typical Goal Times**
$ap1^2_1$, $gp1^2_1$, $gt1^2_1$	$ap1^2_2$, $gp1^2_2$, $gt1^2_2$	$gt1^2_i \approx$ t+0.5 sec
$ap2^2_1$, $gp2^2_1$, $gt2^2_1$	$ap2^2_2$, $gp2^2_2$, $gt2^2_2$	$gt2^2_i \approx$ t+1.0 sec
$ap3^2_1$, $gp3^2_1$, $gt3^2_1$	$ap3^2_2$, $gp3^2_2$, $gt3^2_2$	$gt3^2_i \approx$ t+1.5 sec
$ap4^2_1$, $gp4^2_1$, $gt4^2_1$	$ap4^2_2$, $gp4^2_2$, $gt4^2_2$	$gt4^2_i \approx$ t+2.0 sec
$ap5^2_1$, $gp5^2_1$, $gt5^2_1$	$ap5^2_2$, $gp5^2_2$, $gt5^2_2$	$gt5^2_i \approx$ t+2.5 sec
$ap6^2_1$, $gp6^2_1$, $gt6^2_1$	$ap6^2_2$, $gp6^2_2$, $gt6^2_2$	$gt6^2_i \approx$ t+3.0 sec
$ap7^2_1$, $gp7^2_1$, $gt7^2_1$	$ap7^2_2$, $gp7^2_2$, $gt7^2_2$	$gt7^2_i \approx$ t+3.5 sec
$ap8^2_1$, $gp8^2_1$, $gt8^2_1$	$ap8^2_2$, $gp8^2_2$, $gt8^2_2$	$gt8^2_i \approx$ t+4.0 sec
$ap9^2_1$, $gp9^2_1$, $gt9^2_1$	$ap9^2_2$, $gp9^2_2$, $gt9^2_2$	$gt9^2_i \approx$ t+4.5 sec
$ap10^2_1$, $gp10^2_1$, $gt10^2_1$	$ap10^2_2$, $gp10^2_2$, $gt10^2_2$	$gt10^2_i \approx$ t+5.0 sec

Primitive (level 2)

Commands to Primitive level BG processes would have the form:

Driver Command Structure

ActionCommand = $ac1^2_1$	GoalCommand = $gc1^2_1$	GoalTime = $gt1^2_1 \approx$ t + 0.5 sec
NextActionCommand = $ac2^2_1$	NextGoalCommand = $gc2^2_1$	NextGoalTime = $gt1^2_1 \approx$ t + 1.0 sec

Primitive level plans for the Servo level BG units would have the form:

Velocity Plan	**Goal Times**
$ap1^1_1$, $gp1^1_1$, $gt1^1_1$	$gt1^1_i$ = t+50 ms
$ap2^1_1$, $gp2^1_1$, $gt2^1_1$	$gt2^1_i$ = t+100 ms
$ap3^1_1$, $gp3^1_1$, $gt3^1_1$	$gt3^1_i$ = t+150 ms
$ap4^1_1$, $gp4^1_1$, $gt4^1_1$	$gt4^1_i$ = t+200 ms
$ap5^1_1$, $gp5^1_1$, $gt5^1_1$	$gt5^1_i$ = t+250 ms
$ap6^1_1$, $gp6^1_1$, $gt6^1_1$	$gt6^1_i$ = t+300 ms
$ap7^1_1$, $gp7^1_1$, $gt7^1_1$	$gt7^1_i$ = t+350 ms
$ap8^1_1$, $gp8^1_1$, $gt8^1_1$	$gt8^1_i$ = t+400 ms

$ap9^1_1$, $gp9^1_1$, $gt9^1_1$ $gt9^1_i$ = t+450 ms
$ap10^1_1$, $gp10^1_1$, $gt10^1_1$ $gt10^1_i$ = t+500 ms

Note that time intervals in plans become uniform at the Primitive level and below.

Servo (level 1)

Commands to Servo level BG controllers would have the form:

Velocity Command Structure
ActionCommand = $ac1^1_1$ GoalCommand = $gc1^1_1$ GoalTime = $gt1^1_1$ = t + 50 ms
NextActionCommand = $ac2^1_1$ NextGoalCommand = $gc2^1_1$ NextGoalTime = $gt2^1_1$ = t + 100 ms

Servo level plans for each Actuator would have the form:

Wheel Motors	**Front Steer Motor**	**Rear Steer Motor**	**Goal Times**
$ap1^0_1$, $gp1^0_1$, $gt1^0_1$	$ap1^0_2$, $gp1^0_2$, $gt1^0_2$	$ap1^0_3$, $gp1^0_3$, $gt1^0_3$	$gt1^0_i$ = t+5 ms
$ap2^0_1$, $gp2^0_1$, $gt2^0_1$	$ap2^0_2$, $gp2^0_2$, $gt2^0_2$	$ap2^0_3$, $gp2^0_3$, $gt2^0_3$	$gt2^0_i$ = t+10 ms
$ap3^0_1$, $gp3^0_1$, $gt3^0_1$	$ap3^0_2$, $gp3^0_2$, $gt3^0_2$	$ap3^0_3$, $gp3^0_3$, $gt3^0_3$	$gt3^0_i$ = t+15 ms
$ap4^0_1$, $gp4^0_1$, $gt4^0_1$	$ap4^0_2$, $gp4^0_2$, $gt4^0_2$	$ap4^0_3$, $gp4^0_3$, $gt4^0_3$	$gt4^0_i$ = t+20 ms
$ap5^0_1$, $gp5^0_1$, $gt5^0_1$	$ap5^0_2$, $gp5^0_2$, $gt5^0_2$	$ap5^0_3$, $gp5^0_3$, $gt5^0_3$	$gt5^0_i$ = t+25 ms
$ap6^0_1$, $gp6^0_1$, $gt6^0_1$	$ap6^0_2$, $gp6^0_2$, $gt6^0_2$	$ap6^0_3$, $gp6^0_3$, $gt6^0_3$	$gt6^0_i$ = t+30 ms
$ap7^0_1$, $gp7^0_1$, $gt7^0_1$	$ap7^0_2$, $gp7^0_2$, $gt7^0_2$	$ap7^0_3$, $gp7^0_3$, $gt7^0_3$	$gt7^0_i$ = t+35 ms
$ap8^0_1$, $gp8^0_1$, $gt8^0_1$	$ap8^0_2$, $gp8^0_2$, $gt8^0_2$	$ap8^0_3$, $gp8^0_3$, $gt8^0_3$	$gt8^0_i$ = t+40 ms
$ap9^0_1$, $gp9^0_1$, $gt9^0_1$	$ap9^0_2$, $gp9^0_2$, $gt9^0_2$	$ap9^0_3$, $gp9^0_3$, $gt9^0_3$	$gt9^0_i$ = t+45 ms
$ap10^0_1$, $gp10^0_1$, $gt10^0_1$	$ap10^0_2$, $gp10^0_2$, $gt10^0_2$	$ap10^0_3$, $gp10^0_3$, $gt10^0_3$	$gt10^0_i$ = t+50 ms

Actuators (level 0)

Commands to Actuators would have the form:

Actuator i
ActionCommand^0_i = $ac1^0_i$ GoalCommand = $gc1^0_i$ GoalTime = $gt1^0_i$ = t + 5 ms

Where:
$ac1^0_i = ap1^0_i + kfb(gc1^0_i - x1*^0_i)$
$x1*^0_i$ = predicted state of i-th actuator at next sample
$gc1^0_i = gp1^0_I$
kfb = feedback gain

Example Data Structures

An example of a C++ class data structure for a command from the Vehicle level to the Subsystem Autonomous Mobility level might be:

```
/*  GoToHillCrest ***************************************/
class GO_TO_HILL_CREST_CMD : public RCS_CMD_MSG
{
public:
    GO_TO_HILL_CREST_CMD();          //  Constructor
    void update(CMS *);              //  update function.

                //  action modifiers
    int stealthiness;                //  1 to 100% stealthy
    double  speedLimit;              //  in meters/sec

                //  GoalCommand
    double  x_goal;                  // desired x position on a map about 50 meters away
    double  y_goal;                  // desired y position on a map  about 50 meters away
    char  speedAtGoal;               //  desired speed in m/s at GoalCommand (0 if stop at goal)
    double  headingAtGoal;           //  desired heading at GoalCommand

                //  goal modifiers
    double  timeToGetToGoal;         //  ~ 5 seconds for a vehicle level command
    double  timeTolerance;           //  ± seconds
    double  goalTolerance;           //  close enough radius to goal
};
```

An example of a status message from the Autonomous Mobility Subsystem level Planner to the Vehicle level Executor might be:

```
/*  Status feedback ***************************************/
class AM_VEHICLE-STATUS : public RCS_STAT_MSG
{
public:
    AM_VEHICLE-STATUS ();   // Constructor
    void update(CMS *);              // update function.

    boolean  ExitIfPastGoal;         //  task done flag

                //  predicted state yd* at command GoalTime = gt1[3]1
    double  x_predictedAtGoalTime;
    double  y_predictedAtGoalTime;
    double  speed_predictedAtGoalTime;
    double  heading_predictedAtGoalTime;

                //  estimated time to reach GoalCommand
    double estimatedTimeToGoal;

                //  predicted state at planning horizon (i.e., at last subgoal yd10[2]1)
    double  x_predictedAtPlanHorizon;
    double  y_predictedAtPlanHorizon;
    double  speed_predictedAtPlanHorizon;
    double  heading_predictedAtPlanHorizon;
};
```

3.15 Replanning

Multiple levels of deliberative planning make it possible for plans to be recomputed frequently enough that they never become obsolete. Planners generate new plans well before current plans are fully executed. Typically, replanning is completed by the time the first subgoal is achieved in the current plan (e.g., replanning at level 3 occurs about every 500 milliseconds). Executors react to sensory feedback even faster[5] (e.g., reaction at level 3 occurs within 100 milliseconds.)

To achieve this rate of replanning, it is necessary to limit the amount of data in the world model that needs to be refreshed between each planning cycle. Multilevel representation of space limits the number of resolution elements required in maps and the amount of detail in symbolic data structures at each level. Multilevel representation of time limits the number of events and temporal detail required at each level. The world model in any node is rich and detailed within the region of attention, but contains only the amount of resolution in space and time required for making decisions in that node. This enables the world model in each node to be updated in real-time.

To replan frequently, it is also necessary to limit the amount of search required to generate new plans. There are several ways to limit the search. One is to pre-compute and store plans that can be selected by a rule-based planner in response to the recognition of an object, event, or situation. A second approach is to limit the range and resolution of the state space that needs to be searched and evaluated. At each level, the range and resolution of maps can be limited to less than 64,000 resolution elements.

The 4D/RCS architecture has an interface between deliberative and reactive execution in every node at every hierarchical level. This enables 4D/RCS to fully realize the desirable traits of both deliberative and reactive control in a practical system. Multiple levels of deliberative planning ensure that plans can be recomputed frequently enough that they never become obsolete. Multiple levels of representation cause the planning search space to be limited in range and resolution, and the plans to be limited in the number of steps and amount of detail. Multiple levels of feedback from the environment ensure that reactive behavior can be generated with a minimum of feedback time delay. Table 1 contains suggested 4D/RCS specifications for the planning horizon, replanning interval, and reaction latency at all eight levels:

Level	Planning horizon	Replan interval	Reaction latency
1 Servo	50 ms	50 ms	5 ms
2 Primitive	500 ms	50 ms	50 ms
3 Subsystem	5 s	500 ms	200 ms
4 Vehicle	50 s	5 s	500 ms
5 Section	10 min	50 s	2 s
6 Platoon	1 h	5 min	5 s
7 Company	5 h	30 min	10 s
8 Battalion	24 h	2 h	20 s

Table 1. The Planning Horizon, Replan Interval, and Executor Reaction Latency at each level of the 4D/RCS hierarchy

[5] Except at level where replanning and reaction times are the same. At levels 2 and above, the difference between replanning and reacting becomes more significant with each successively higher level.

The planning horizon refers to the future point in time to which each level plans. Plans at each level typically have 5 to 10 steps between the anticipated starting state and a planned goal state at the planning horizon. Thus, the planning horizon typically grows about one order of magnitude longer at each successively higher level.

Reaction latency is the minimum delay through the reactive feedback loop at each level. Reaction can interrupt cyclic replanning to immediately select an emergency plan, and to begin a new replanning cycle based on new information. Reaction latencies at each level are determined by computational delays in updating the world model as well as the sampling frequency and computation cycle rate of the Executors. The fastest servo update rate on a typical vehicle controller is 200 Hz. Thus, the reaction latency at the Servo level is 5 ms. The required execution cycle rate at other levels depends on the dynamics of the mechanism being controlled and the speed of the available computers.

3.16 Two Kinds of Plans

There are two kinds of plans that are required by the FCS vehicles: (1) path plans for mobility, and (2) task plans for tactical behaviors. A typical path plan consists of a series of waypoints on a map. A typical task plan consists of a set of instructions or rules that describes a sequence of actions and subgoals required to complete the task. Both path plans and task plans can be represented in the form of augmented state graphs, or state tables, which define a series of planned actions (subtasks) with a desired state (subgoal) to be achieved by each action in the plan. Typically states are represented by nodes, and actions are represented by arcs that connect the nodes. Both types of plans can be executed by the same executor mechanism.

In principle, both types of planning can be performed by searching the space of possible futures to find a desirable solution. However, path planning typically requires searching only a two-dimensional space on a map. Task planning requires searching an N-dimensional space of all possible states and actions. Searching high dimensional spaces can be accomplished by evolutionary algorithms [Fogel99] or reinforcement learning techniques. [Sutton and Barto98] However, these methods are typically too slow for real-time use at levels where plans must be recomputed faster than once every few minutes. Therefore, real-time task planning is typically done by searching a library of schema or recipes that have been developed off-line and stored where they can be accessed by rules or case statements when conditions arise. When there are more than one recipe or schema that are appropriate to a task, each may be submitted to the world model for simulation and the predicted results evaluated by the value judgment process. The plan selector then selects the best recipe or schema for execution.

In 4D/RCS, path planners use cost maps that represent the estimated cost or risk of being in, or traversing, regions on the map. Values represented in cost maps depend on mission priorities and knowledge of the tactical situation represented in the KD. Path planners search the cost maps for routes that have the lowest cost under a given situation. Task planners use rules of engagement, military doctrine, and case-based reasoning to select modes of operation and

schema for tactical behaviors. State variables such as mission priorities and situational awareness determine cost functions and hence decisions regarding which type of behavior to select.

For example, if enemy contact is likely or has occurred, cost maps of open regions and roads will carry a high cost. Regions near tree lines and under tree cover will have lower cost as long as they are cleared of enemy threats. In this case, path planners will plan cautious routes near tree lines or through wooded areas, and task planners will plan behaviors designed to search for evidence of enemy activity in likely places. However, if enemy contact is unlikely, roads will have a very low cost and open regions will carry a lower cost than wooded areas. This will cause path planners to plan higher speed routes on the road or through open regions, and task planners to focus on issues such as avoiding local traffic. Thus, a very small amount of information, such as knowledge that enemy contact is likely or unlikely, can completely change the tactical behavior of the vehicle in a very logical, intuitive, and meaningful way.

For the Demo III program, the range and resolution of maps is limited to about 128 resolution elements from the center of the map in each direction at each level. This means that each map contains about 256x256 ($\approx$ 64,000 pixels). The suggested range and resolution of maps at all levels of the FCS 4D/RCS hierarchy are shown in Table 2. Maps at each level provide information to planners about the position, attributes, and class of entities. For example, maps at various levels may indicate the shape, size, class, and motion of objects such as obstacles and vehicles, and the location of roads, intersections, bridges, streams, woods, lakes, buildings, and towns.

Level	Map Resolution	Map Range	Function Performed
1 Servo	n/a	n/a	Actuator servo
2 Primitive	4 cm	5 m	Vehicle heading, speed
3 Subsystem	40 cm	50 m	Obstacle avoidance
4 Vehicle	4 m	500 m	Single vehicle tactical behaviors
5 Section	30 m	2 km	Section level tactical behaviors
6 Platoon	30 m	10 km	Platoon level tactical behaviors
7 Company	30 m	30 km	Company level tactical behaviors
8 Battalion	30 m	100 km	Battalion level tactical behaviors

Table 2. Range and resolution of maps at all levels in the proposed FCS 4D/RCS architecture. Range is measured from the vehicle at the center of each map.

In general, map range and resolution depend on velocity and the planning time horizon. For any given planning time horizon, the map range must be sufficient to guarantee that the plan will fit on the map. For different vehicle speeds, the map resolution required for planning at various levels will be different. The numbers in Table 2 are for ground vehicles traveling about 10 m/s. A helicopter skimming over the ground at 100 m/s would require planning maps with an order of magnitude greater range and an order of magnitude lower resolution than that shown above. For systems with widely varying velocities, map range and resolution may need to be velocity dependent.

4.0 ENGINEERING GUIDELINES

The remainder of this document describes engineering guidelines for building systems based on the 4D/RCS reference model architecture. At this point we will drop one layer deeper into the 4D/RCS methodology and suggest a specific approach to designing a set of software modules that can implement the functions of planning, control, reflex response, reasoning, world modeling, simulation, visualization, recursive estimation, grouping, computation of attributes, classification, and establishment of relationships.

In this section, we will discuss a method that could be used to implement the BG, WM, SP, and VJ processes. And we will outline a set of algorithmic approaches that could be used to build practical systems that conform to the 4D/RCS reference model in a comprehensive manner. We do not pretend that these guidelines describe the only possible method for implementing 4D/RCS systems, or even that they represent the best possible design. Rather, these engineering guidelines are offered:

First, to demonstrate that at least one possible approach exists to implement the 4D/RCS reference model; and

Second, to clarify the engineering issues that must be addressed if attempts to build intelligent machines that approach human levels of performance in responsiveness, dexterity, adaptability, reliability, and cognitive understanding are to be successful.

A number of practitioners have applied these and other methods that focus on particular aspects of the 4D/RCS architecture. A.J. Barbera and M.L Fitzgerald have applied a task decomposition based RCS approach to many industrial automatic systems [ATR 02]. Kevin Passino and colleagues describe an implementation method based on a RCS software library [Gazi et al. 01]. Huang and colleagues have investigated the combination of the object-oriented software engineering paradigm and the task based RCS methods [Huang et al. 00 and Huang and Messina 96]. Messina, Horst and colleagues have investigated a component based method [Horst 00 and Messina 99]. Evans, Messina and colleagues focus on knowledge engineering [Evans et al. 02].

We begin our discussion of engineering guidelines with the process of behavior generation.

4.1 Behavior Generation

A BG process resides within RCS_NODE that represents an operational unit in an organizational hierarchy. The BG process accepts tasks and plans and executes behavior designed to accomplish those tasks. The internal structure of the BG process consists of a planner and a set of executors (EX) as shown in Figure 9. At the upper right, task commands from a supervisor are input to a BG process. A planner module decomposes each task into a set of coordinated plans for subordinate BG processes. This is accomplished by the following search process:

While the planning period is open

{

1) the BG planner hypothesizes a tentative_plan;

2) the WM predicts the probable_result of the tentative_plan;

3) the VJ evaluates the probable_result_value;

4) a plan selector within the BG planner checks to see if the probable_result_value is greater than the previous probable_result_value of the plan already in the current_best_plan_buffer,

{

if it is, then the tentative_plan replaces the current plan in the current_best_plan_buffer;
else continue;

}

};

On the next execution clock cycle,

1) Move the contents of the current_best_plan_buffer into the executor_plan_buffers;
2) Begin replanning immediately, or wait until the next planning cycle triggers;

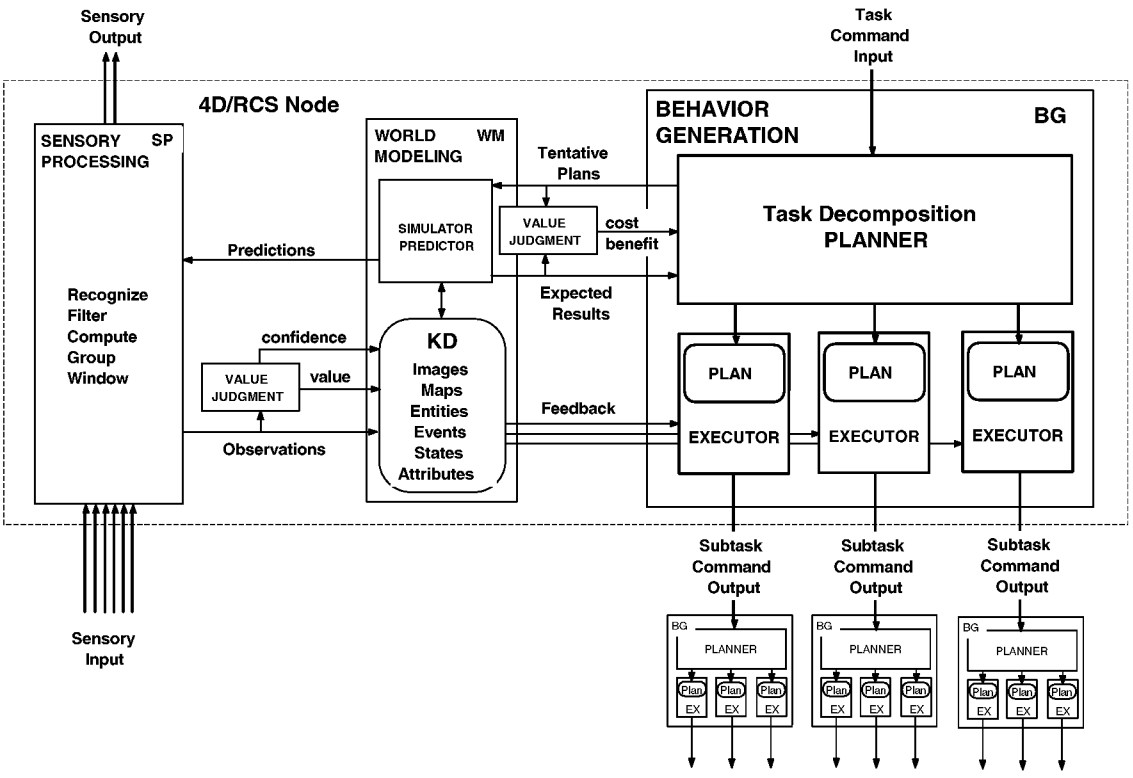


Figure 9. A typical 4D/RCS computational node, RCS_NODE. Task command inputs come from a higher level behavior generation (BG) process in the 4D/RCS hierarchy. Each input task command is decomposed into a plan consisting of subtasks for subordinate BG processes. A world modeling (WM) process maintains a knowledge database (KD) that is the BG unit's best estimate of the external world. A sensory processing (SP) system operates on input from sensors by focusing attention (i.e., windowing), grouping, computing attributes, filtering, and recognizing entities, events, and situations. A value judgment (VJ) process evaluates expected results of tentative plans. A VJ process also evaluates entities, events, and situations entered into the KD.

The executor_plan_buffer forms an interface between the planner and executor processes. This is a critical interface between the deliberative and reactive processes in the intelligent controller. The executor_plan_buffer enables the planning process to run asynchronously and at a different rate from the execution cycle of the reactive control loop. As plans are developed, they are loaded into the executor_plan_buffers. The planner is free to run on its own so long at the executor_plan_buffers are kept supplied with a current_best_plan. In the 4D/RCS architecture this interface between planners and executors exists at every level in the computational hierarchy. Thus, the interface between deliberative and reactive processes is not localized to a particular level, but is distributed throughout the 4D/RCS architecture

An executor services each subordinate BG unit, issuing subtask commands, monitoring progress, compensating for errors and differences between planned and observed situations in the world, and reacting quickly to emergency conditions with reflexive actions. The Executors use feedback via a real-time knowledge database KD to generate reactive behavior. Predictive capabilities in the WM can enable preemptive behavior.

SP and WM processes interact to support windowing, grouping, recursive estimation, and classification. WM updates the KD with images, maps, entities, events, attributes, and states. These enable deliberative, reactive, and preemptive behavior. Coordination between subordinate BG processes is achieved by cross-coupling among plans and sharing of information among Executors via the KD. The planner in Figure 9 can be further decomposed into three subprocesses:

1. A Job Assignor (JA)
2. A set of SChedulers (SC)
3. A Plan Selector (PS)

as shown in Figure 10. The JA subprocess acts as a supervisor of the BG unit. Each pair of SC and EX subprocesses act as peer subordinate agents within the BG unit. The JA supervisor subprocess interacts with the SC subprocesses to make plans that the respective EX subprocesses execute. Plans consist of planned actions and desired results. Each EX subprocess merges its plan with feedback from the world model to generate plan-driven reactive behavior. Each EX sends commands to, and monitors the status of, its respective subordinate BG unit.

The set of subprocesses JA, SC, EX can be grouped in two ways: 1) as organizational units, or 2) as agents. An organizational unit grouping consists of a JA subprocess as supervisor of the BG unit, with SC/EX pairs as staff officers within the unit. An agent grouping consists of SC/EX/JA triplets working together as agents, where each agent is a subordinate in the higher level unit, and supervises a lower level unit.

The set of intelligent agents that interact within the BG process may correspond to a team of persons interacting within an organizational unit or a set of coordinated processes interacting within an intelligent control system.

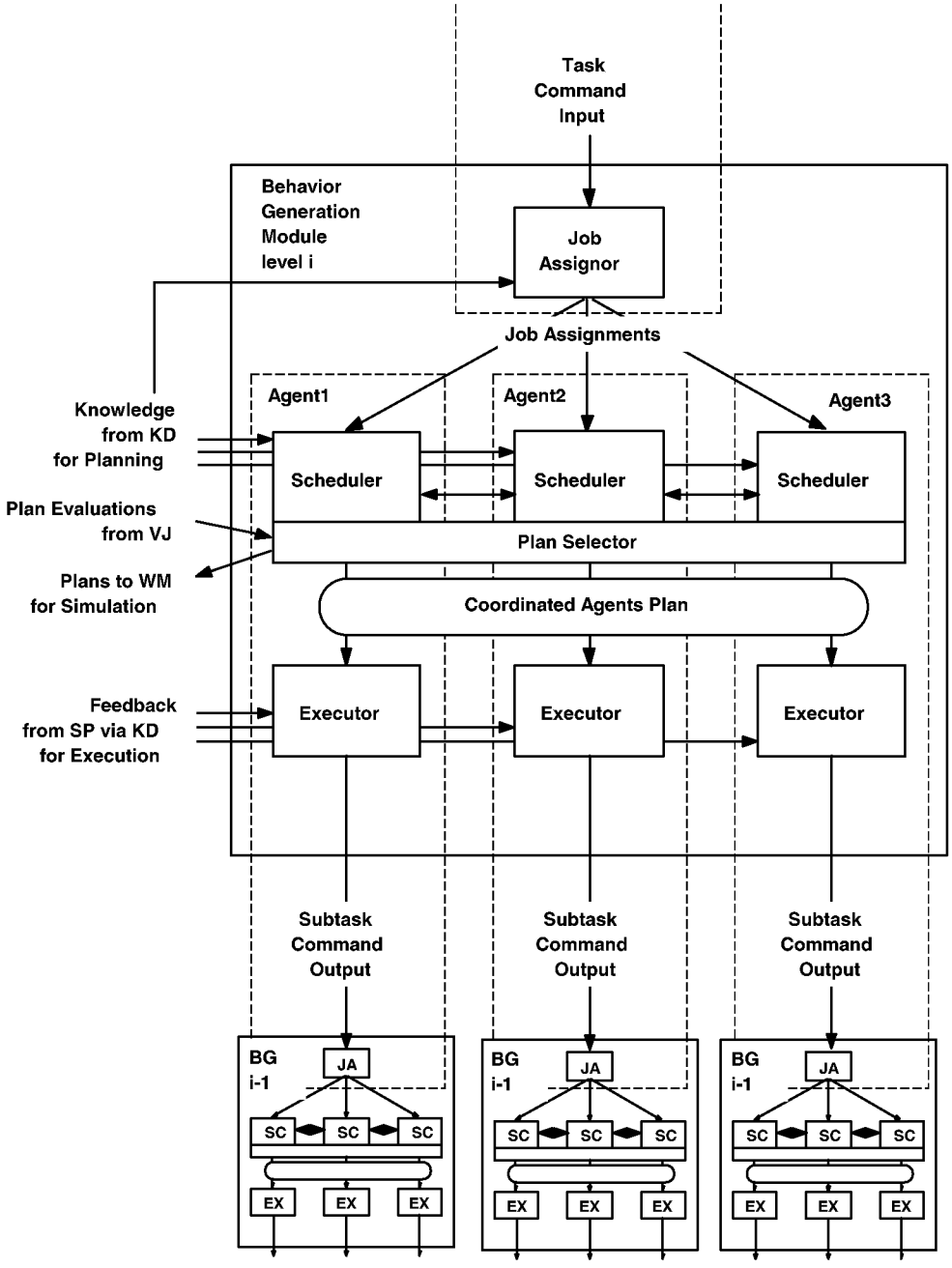


Figure 10. Internal structure of Behavior Generation (BG) processes at two levels with three agents at each level.

Df. The **Job Assignor (JA)** is *a subprocess of BG that decomposes input tasks into job assignments for each agent within the BG process.*

The Job Assignor (JA) performs four functions:

1) JA accepts input task commands from an Executor in a higher level BG process. Task commands typically involve at least two steps in the current higher level plan.

2) JA decomposes the commanded input task into a set of jobs that are assigned to agents within the BG process[6].

3) JA transforms each job assignment into a coordinate frame of reference which is appropriate for the performing agent.

4) JA allocates resources to the agents to enable them to accomplish their assigned jobs.

The JA subprocess functions as the supervisor of the peer agents (each of which consists of a scheduler, executor, and job assignor of the next lower level) within the BG operational unit.

Df. The **SCheduler (SC)** is *a subprocess within BG that accepts a job assignment and computes a schedule for the EX subprocess with which it is paired.*

There is a SC subprocess for each EX subprocess within the BG process. Each SC accepts a job assignment from the JA. Each SC computes a sequence of activities that accomplishes the assigned job. The set of SC subprocesses within the BG process may need to communicate with each other to coordinate their sequences of planned actions and subgoals for their respective EX subprocesses in the same BG process, or even possibly with EX subprocesses in other BG processes. A SC subprocesses may negotiate with the JA subprocess or with its peer SC subprocesses regarding shared resources, to resolve conflicts and optimize coordination of actions among cooperating EX subprocesses.

Within the 4D/RCS methodology, a system designer can implement different management styles by different distribution of duties and responsibilities to the JA and SC subprocesses within a BG process. For example, an autocratic or micro-management style can be achieved by giving both job assignment and most scheduling responsibilities to the JA subprocess of the supervisor agent, leaving very little discretion to the SC subprocesses of the subordinate agents. On the other hand, a market negotiation management style can be achieved by giving the JA subprocess only general oversight responsibilities and allowing the subordinate SCs to negotiate among themselves for job assignments and resources. This gives responsibility to the subordinates for scheduling their own activities. In either case, the JA and SC subprocesses together have responsibility for producing a plan that can be followed by the EX subprocesses to accomplish the commanded task.

The subprocess that is responsible for selecting the tentative plan with the best evaluation is the Plan Selector.

---

[6] At the vehicle level and below, the assignment of jobs and resources to agents may be fixed by the system design. At higher levels, the JA subprocess may need to take into consideration the reconfiguration of the BG command and control hierarchy. From time to time, an agent with its subordinate BG unit may be added to or subtracted from a higher level BG unit, or transferred from one higher level unit to another. For example, a flight of strike aircraft may be under the control of the home base flight controller during take-off and landing, transfer to the control of a forward air controller at the point of attack, and be transferred to the control of intermediate controllers on the way to and from the target area. If added, the new agent joins the higher level BG unit as a subordinate.

Df. The **Plan Selector (PS)** is *a subprocess of BG that works with a WM plan simulator and VJ plan evaluator to select the best overall plan (i.e. job assignment, resource allocation, and coordinated schedule) for execution by the EX subprocesses in the BG process.*

The Plan Selector maintains the plan buffer used by the EX subprocesses and assures that the EXs always have the latest and best plan currently available.

Df. The **EXecutor (EX)** is *a subprocess within BG that executes its portion of the selected plan, coordinating actions when required, and correcting for errors between planned results and the evolution of the world state reported by the world model.*

There is an EX subprocess for each subordinate BG unit. Each EX subprocess closes a feedback control loop through the RCS_NODE to which it belongs. Each EX subprocess detects errors between its current planned subgoal (i.e., desired state) and the observed (or predicted) state of the world, and issues output subcommands designed to null the errors. The EX subprocess may also output the current planned action as a feed forward command. The EX subprocess detects when goals are achieved and increments to the next task and goal in the plan. The EX subprocess may also detect when limits have been exceeded, or when emergency flags have been raised, and take immediate emergency action.

4.1.1 Tasks and Task Decomposition

A task consists of an activity and a goal. The activity is designed to achieve the goal, and the goal is a desired state to be achieved or maintained.

There are two types of task: one where the goal is to *achieve* a desired state, and a second where the goal is to maintain a desired state. For type 1 tasks, achieving the goal terminates the task. For example, a type 1 task may be <to drive to a location>, <to assemble an engine>, or <to shop for groceries>. Once the goal is achieved, the task is completed, and the next task can begin. For type 2 tasks, the objective is to maintain the task goal state until a different task is commanded. For example, a type 2 task may be <to maintain a temperature>, <to maintain a velocity>, or <to maintain an altitude>.

Both types of task imply action or purposeful lack of action. In natural language, action is demoted by *verbs*. In the above examples of tasks, the infinitive form of the verb is used (e.g. task = <to drive to a location>). The application of the verb <do> to the <task> transforms the task verb from its infinitive to its active form. It puts the task verb into the form of a command. For example, <do_<to look for enemy tanks>> becomes a command to <look for enemy tanks>, where <look for> is the action verb, and <enemy tanks> is the *object* upon which the action is done.

For any task, there is an action verb that is a command to perform the task. The set of tasks that a BG process is capable of performing, therefore, defines a vocabulary of action verbs or commands that the BG process is capable of accepting.

A task such as < drive vehicle to location x>, may be performed by a number of agents (drivers) on a variety of objects (vehicles). A task such as "assemble an engine" may be performed by a number of agents (assembly line workers) on a number of objects (e.g., pistons, bearings, gaskets, head, and oil pan). At the input of the BG process, a task command such as <look for tanks> or <assemble an engine> is encoded as a single command to do a single activity to achieve a single goal. The JA subprocess at the input of the BG process performs the job assignment function of decomposing the task into a set of jobs for a set of agents (e.g., a suite of RSTA sensors, or team of assembly line workers). For each agent, a SC subprocess schedules a sequence of subtasks, which are executed by the respective EX subprocess so as to accomplish the commanded task.

Df. A **task goal** is the *desired result that a task performed by a BG process is to achieve or maintain.*

A task command can be represented as an imperative sentence in a message language that directs a BG process to do a task, and specifies the object(s), the task goal, and the task parameters. A task command has a specific interface defined by a task command frame.

Df. The **interface between BG processes at level(i+1) and level(i)** *consists of a message channel that encodes task command frames, and a message channel that encodes status responses that return to level(i+1) either directly, or through the world model.*

Df. A **task command frame** is *a data structure containing the information necessary for commanding a BG process to do a task.*

A **task command frame** includes:

- ☐ **Task name** (from the vocabulary of tasks the receiving BG process can perform)
- ☐ **Task identifier** (unique for each commanded task)
- ☐ **Task goal** (a desired state to be achieved or maintained by the task)
- ☐ **Task object**(s) (on which the task is to be performed)
- ☐ **Task parameter(s)** (such as speed, force, priority, constraints, coordination requirements)
- ☐ **Planned next task** (for look-ahead beyond the current commanded task)

The planned next task allows the local BG planner to look ahead to its planning horizon, even when that planning horizon extends beyond the end of its current task. In some implementations, the task command frame includes the entire plan from the higher level plan. However, if the system is designed right, it is seldom necessary for a planner to look beyond the planned next task in the higher level plan. If information about the more distant future is required at the lower level, this information should be explicitly provided by the upper level to

the lower level in the form of a "get ready" command. For example, if a heater requires a heat-up period that is longer than the lower level planning horizon, the higher level planner should plan for a specific command to be sent to the lower level BG process to turn on the heater at the proper pre-heat time.

For any task to be successfully accomplished, the intelligent system must possess task knowledge that describes how to do the task. Conversely, for any BG process, there must exist a library of tasks that the BG process knows how to do.

Task knowledge consists of the skills and abilities required for performing tasks. For example, task knowledge may describe how to steer around an obstacle, how to follow a road, how to navigate from one map location to another, how to avoid being observed from a particular location, how to cross a creek or gully, how to encode a message, or how to load, aim, and fire a weapon. Task knowledge may also include a list of equipment, materials, and information required to perform a task, a set of conditions required to begin or continue a task, a set of constraints on the operations that must be performed during the task, and a set of information, procedures, and skills required to successfully execute the task, including error correction procedures, control laws, and rules that describe how to respond to failure conditions.

Task knowledge may be acquired through learning, or inherited through instinct. It may be provided in the form of schema or algorithms designed by a programmer, or discovered by heuristic search over the space of possible behaviors. Task knowledge, together with information supplied by task commands, is what enables the Behavior Generation processes to perform tasks. Task knowledge can be represented in a task frame.

Df. A **task frame** is *a data structure specifying all the information necessary for accomplishing a task.*

A task frame is essentially a recipe that specifies the materials, tools, and procedures for accomplishing a task. A task frame may include:

- **Task name** (from the library of tasks the system knows how to perform). The task name is a pointer or an address in a database where the task frame can be found.
- **Task identifier** (unique id for each task commanded). The task identifier provides a method for keeping track of tasks in a queue.
- **Task goal** (a desired state to be achieved or maintained by the task). The task goal is the desired result to be achieved from executing the task.
- **Task objects** (on which the task is to be performed). Examples of task objects include targets to be attacked, objects to be observed, sectors to be reconnoitered, vehicles to be driven, weapons or cameras to be pointed.
- **Task parameters** (that specify, or modulate, how the task should be performed). Examples of task parameters are priority, speed, time of arrival, and level of aggressiveness.
- **Agents** (that are responsible for executing the task). Agents are the subsystems and actuators that carry out the task.
- **Task requirements** (tools, resources, conditions, state information). Tools may include sensors and weapons. Resources may include fuel and ammunition.

Conditions may include weather, visibility, soil conditions, daylight or darkness. State information may include the position and readiness of friendly forces and other vehicles, position and activity of enemy forces, and stage in the battle.

- **Task constraints** (upon the performance of the task). Task constraints may include visibility of regions of the battlefield, timing of movements, requirements for covering fire, phase lines, and sector boundaries.
- **Task procedures** (plans for accomplishing the task, or procedures for generating plans). There is, typically, a library of plans for various routine contingencies and procedures that specify what to do under various kinds of routine and unexpected circumstances.
- **Control laws and error correction procedures** (defining what action should be taken for various combinations of commands and feedback conditions). These may be developed during system design, refined during testing, and possibly modified through learning from experience.

Note that many of the items in the task frame are supplied by the task command frame. Other items are provided by a priori knowledge.

Df. A **task object** is a *thing in the world that is acted upon by an agent to accomplish a task.*

For any task object, there can be assigned a word (noun) that is the name of the object upon which the task is performed.

Df. **Task parameters** are *modifiers that specify when, where, how, or why a task should be performed, or that prioritize a task, or specify how much risk or cost should be accepted in performing the task, or set constraints, or modulate the manner in which a task should be performed.*

As defined in section 3, task decomposition is a process by which a task given to a BG process at one level (i) is decomposed into a set of sequences of subtasks to be given to a set of subordinate BG processes at the next lower level (i-1).

The set of task frames that are available to each BG process defines a set of task commands that the BG process can accept. If the BG process is given a command to do a task that it is capable of performing, it uses the knowledge in the corresponding task frame to accomplish the task. If it is given a command to do a task that it is incapable of performing, (i.e. for which there is no task knowledge stored in a task frame, or for which the resources specified in the task frame are not available, or for which conditions specified are not true) it responds to its supervisor with an error message = <can't do because of condition xyz>.

The JA subprocess of the BG process accepts task commands with goals and priorities. The JA, SC, and PS subprocesses develop or select a plan for each commanded task by using a priori task knowledge, heuristics, and value judgment functions combined with real-time information and simulation capabilities provided by world modeling functions. The planning process generates the best assignment of tools and resources to agents, and finds the best

schedule of actions (i.e., the most efficient plan to get from an anticipated starting state to a goal state). EX subprocesses control action by stepping through the plan. At each control cycle, the EX subprocesses issue feed forward commands and compare current state feedback with the desired state specified by the plan. The EX subprocesses merge feed forward actions with feedback error compensation and issue task commands to the next lower level BG processes that maximize the likelihood of achieving the higher level task goal despite noise and perturbations in the system.

Task decomposition consists of six generic functions:

1) A **job assignment function** is performed by a JA subprocess whereby:
   a) A task is divided into jobs to be performed by agents
   b) Resources are allocated to agents
   c) The coordinate frame in which the jobs are described is specified

2) **Scheduling functions** are performed by SC subprocesses whereby a job for each agent is decomposed into a sequence of subtasks (possibly coordinated with other sequences of subtasks for other agents).

3) **Plan simulation** functions are performed by a WM process whereby the results of various alternative plans are predicted.

4) **Evaluation functions** are performed by a VJ process on the results of simulations using a cost/benefit formula.

5) A **plan selection function** is performed by a PS subprocess in selecting the "best" of the various alternative plans for execution. The cost model in the value judgement process is used to determine the best plan from cycle to cycle.

6) **Execution of the best plan** generated by functions 1-5 is performed by EX subprocesses. The plan consists of a set of desired actions and a set of desired resulting states, or subgoals. The set of desired actions form a set of feed forward commands. The set of desired subgoals form a set of desired states. These are compared with observed states, and differences are treated as error signals that are submitted to a control law to compute feedback compensation. Feed forward and feedback compensation are combined to produce sequences of subtask commands to BG processes at the next lower level in the control hierarchy.

The result of task decomposition is that each task (represented by a single task command) presented to a single BG process generates a behavior defined by a set of subtask command sequences presented to a set of subordinate BG processes.

Df. A **job** is an *activity assigned by the Job Assignor to the Scheduler of an agent within a BG process.*

A **job** is the output of a Job Assignor, and the input to the SC subprocess of an agent. A job consists of a description of the work to be done by an agent, the job goal, the resources and

tools necessary to carry out the job, and whatever constraints and requirements that exist for cooperation with other agents. In the 4D/RCS reference architecture, the difference between a job and a task is that a task is something to be performed by a BG process, and a job is something to be performed by an agent. A job can have a job goal, a job command, and a job command frame that are entirely analogous to a task goal, task command, and task command frame.

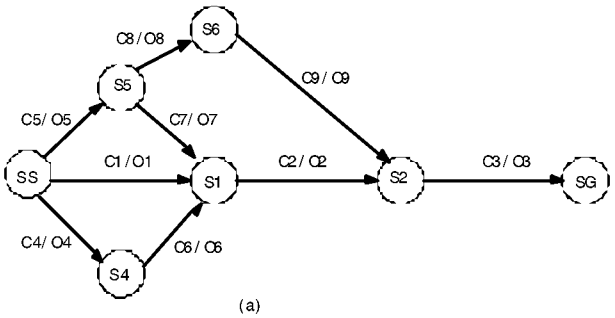
4.1.2 Plans and Planning

As defined in section 3, a plan is a set of subtasks and subgoals that are designed to accomplish a task or job, or a sequence (or set of sequences for a set of agents) of subtasks intended to generate a sequence of subgoals leading from the starting state to the goal state.

In general, a plan is a path through state-space from an anticipated starting state to a goal state. Typically, this path consists of a string (or set of strings) of actions and a string (or set of strings) of resulting states. The actions and resulting states are subtasks. A plan performed by a single agent is a job plan. A plan performed by a set of (possibly coordinated) agents is a task plan.

A plan may be represented declaratively as an augmented state graph, a PERT (Program Evaluation and Review Technique) chart, a Gantt chart, a PSL (Plan Specification Language) [Schlenoff 00] code segment, or a set of reference trajectories through state space. A plan can also be represented procedurally as a program that generates a set of sequences of subtask commands and desired states that generate a path through state space from a start state to a goal state while using resources and satisfying constraints.

Figure 11 illustrates a plan expressed as a state graph where nodes represent desired states (or subgoals) and edges represent actions that cause state changes (or subtasks). Plans can also be expressed as state tables or sets of IF/THEN statements as illustrated in Figure 11. For any state graph, a state table can be constructed such that each edge in the state graph corresponds to a line in the state table, and each node in the state graph corresponds to a state in the state table. The state table form has the advantage that it can be expressed as a set of IF/THEN case statements that can be directly executed in a computer program.



(a)

IF		THEN	
State	Feedback	Next State	Output
SS	C1	S1	O1
SS	C5	S5	O5
SS	C4	S4	O4
S1	C2	S2	O2
S5	C7	S1	O7
S5	C8	S6	O8
S6	C9	S2	O9
S2	C3	SG	O3

(b)

Figure 11. Two forms of plan representations. (a) Shows an example of a state graph representation of a plan that leads from starting state SS to goal state SG. **(b)** Shows the state table that is the dual of the state graph in **(a).**

Figure 12 illustrates the decomposition of a task into a task plan consisting of three job plans for three agents. Each job plan can be represented as a state graph where subgoals are nodes and subtasks are edges that lead from one subgoal to the next. A job plan can also be represented as a set of waypoints on a map, such that the waypoints are subgoals and the paths between waypoints are subtasks.

The plan shown in Figure 12 consists of three linear lists of subtasks and subgoals. In practice, each job plan could have a number of branching conditions that represent alternative actions that may be triggered by situations or events detected by sensors. For example, each job plan might contain emergency action sequences that would be triggered by out-of-range conditions.

Coordination between agents can be achieved by making state transitions in the state graph of one agent conditional upon states or transitions in states of other agents, or by making state transitions of coordinated agents dependent on states of the world reported in the world model.

Df. A **schedule** is *the timing or ordering specifications for a plan.*

A schedule can be represented as a time (or event) labeled sequence of activities or events.

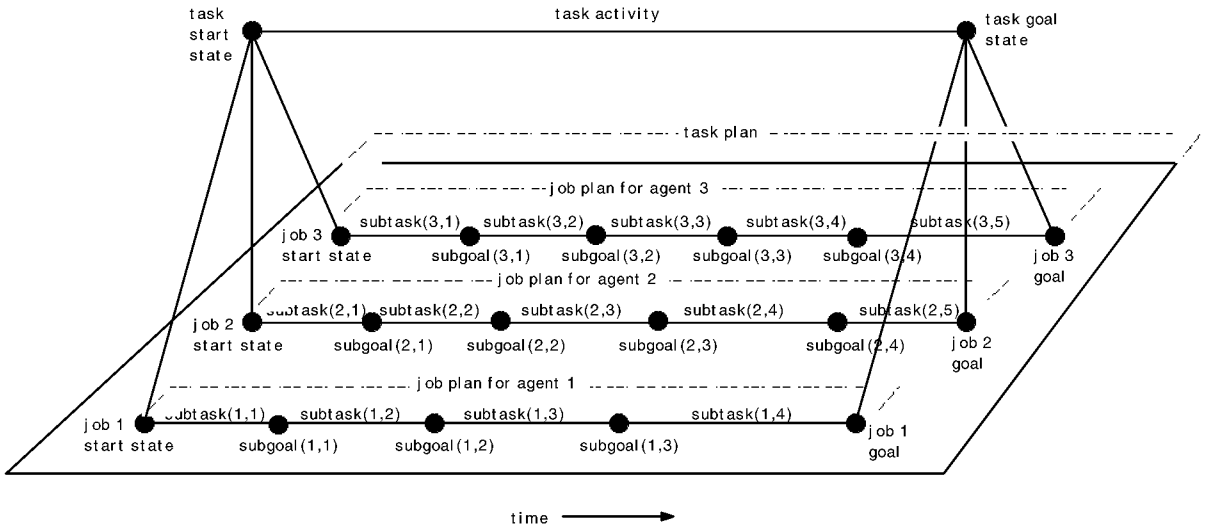


Figure 12. A task plan for three agents. The task is decomposed into three jobs for three agents. Each job consists of a string of subtasks and subgoals. The entire set of subtasks and subgoals is a task plan for accomplishing the task. In this example, a task plan consists of three parallel job plans.

Planning is a process of generating and/or selecting a plan to accomplish a task or job. Task planning is performed by an organizational unit consisting of a number of agents. Task planning consists of the following elements:

(1) **Assigning** jobs to agents and transforming coordinates where required.
(2) **Allocating** resources to agents.
(3) **Generating** a tentative set of concurrent and possibly coordinated job plans, one for each of the agents.
(4) **Simulating** the likely results of this tentative set of job plans.
(5) **Evaluating** the cost/benefit value of each likely result.
(6) **Selecting** the set of job plans with the best evaluation as a task plan to be executed.

A diagram of a typical task planning process is shown in Figure 13. Planning elements (1) and (2) in the above list are performed by JA, which assigns jobs and resources to agents and transforms coordinates from task to job coordinates (e.g. from way point coordinates to steering coordinates, or from map coordinates to camera image coordinates). Element (3) is performed by SC subprocesses, which compute a temporal schedule of subtasks (or subgoals) for each agent and coordinate the schedules between cooperating agents. Element (4) is where a hypothesized string of actions (or subgoals) is submitted to the world model. In the case where hypothesized

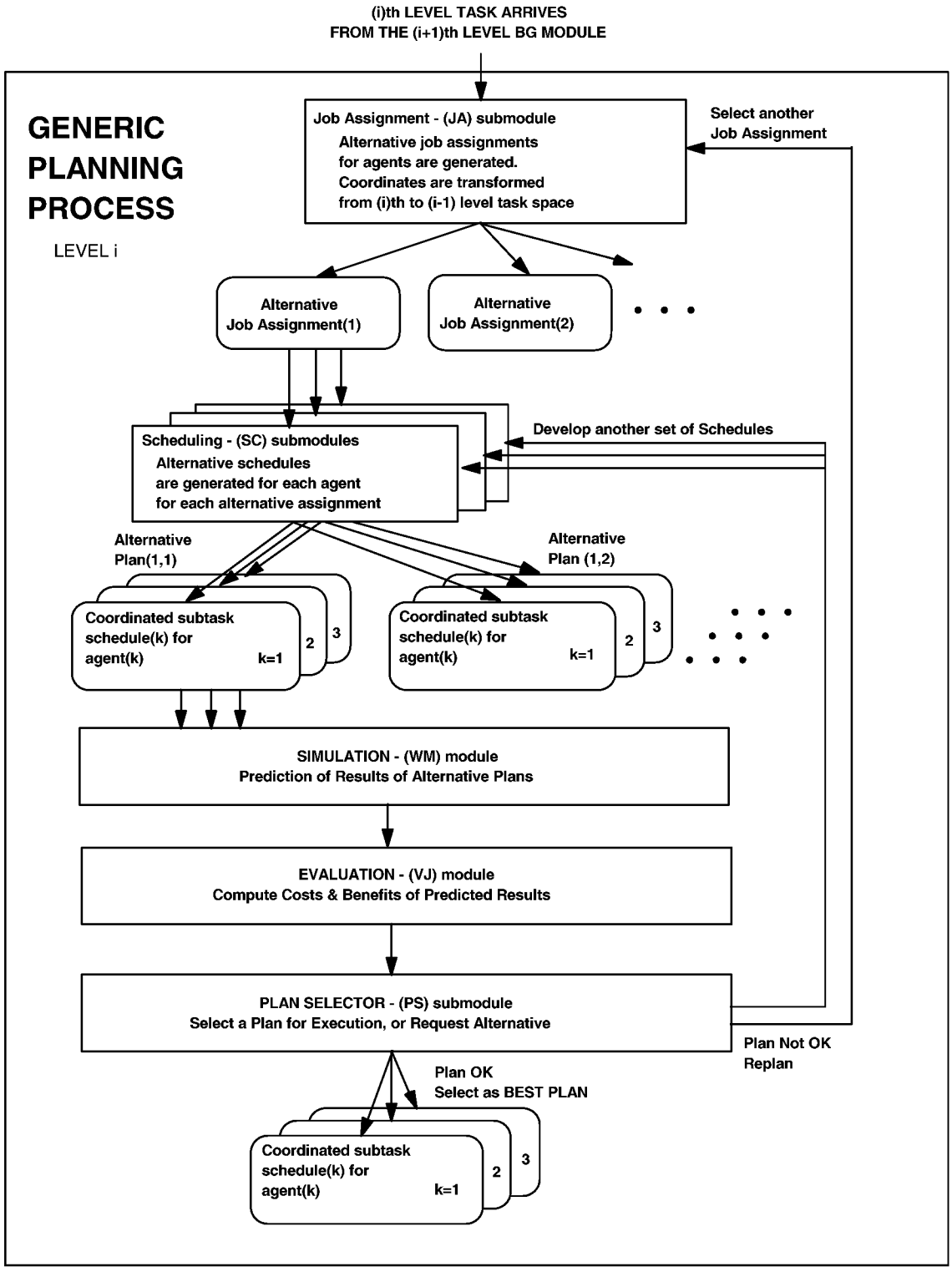


Figure 13. A diagram of planning operations within a 4D/RCS node. Boxes with square corners represent computational functions. Boxes with rounded corners represent data structures.

actions are submitted, the world model simulator uses a model of the environment to generate expected results. In the case where desired subgoals are submitted, the world model uses an inverse model of the environment to compute the feed forward actions required to produce those subgoals. Element (5) is where the Value Judgment (VJ) process applies a cost function to the string of actions and resulting subgoals to determine which of the various actions are submitted, the world model simulator uses a model of the environment to generate expected results. In the case where desired subgoals are submitted, the world model hypothesized alternatives is the best (i.e., most efficient, most effective, lowest cost, or highest payoff). Element (6) is where the evaluations produced by the VJ process are returned to the Plan Selector (PS) subprocess for a decision as to the best plan of action. The hypothesized plan (i.e. set of job plans) with the best evaluation is entered into the set of plan buffers in the EXecutors for execution.

Job planning is performed by a single agent. Job planning consists of the following elements:

1. **Generating** a tentative schedule of activities for a single agent, possibly coordinated with the schedules of other agents.
2. **Simulating** the likely results of each tentative schedule of activities.
3. **Evaluating** the cost/benefit value of each likely result.
4. **Selecting** the tentative schedule with the best evaluation as a job plan to be executed.

Current military practice is typically that planning is done manually using maps, charts, sand tables, and intelligence reports. Plan alternatives are chosen at execution time by rules and human judgment. Plans may be rehearsed by walk-through, or by moving models on a sand table. On the battlefield, plans are sometimes good only until the first shot is fired. Plans typically need to be modified often and quickly to adapt to the flow of battle, to take advantage of openings in the enemy defense, to compensate for losses, or to deal with unanticipated events. Real-time modifications in plans are made by ad hoc methods that depend on the training, experience, and intuition of officers and soldiers in the field.

Military planning is typically a distributed process that is done by many different planners working in many different places at different times. For example, strategic planning typically is done at a higher level than tactical planning and well before the battle begins. As planning is done at successively lower level, each operational unit is required to develop its own plans, scaled to its own time horizon and region of terrain, based on the plans developed at levels above, and constrained by the resources available and conditions on the ground.

Yet, regardless of how, where, or when planning is done, in each stage of planning, the elements of planning listed above typically come into play. However plans are synthesized, a plan eventually must specify the decomposition of tasks for operational units into sets of job assignments for agents, an allocation of resources, and a schedule of subtasks for each agent ordered along the time line. Planning may need to be done iteratively and hierarchically through many levels and by many different agents in different organizations before the plans are ready for execution.

In all cases, plans must be synthesized prior to execution. In highly structured and predictable environments, plans may be computed off-line long before execution. For example,

in manufacturing, completely pre-computed plans, such as NC machine tool code, can be developed months or years before execution. In such cases, real-time planning is unnecessary, and execution consists of simply of loading the appropriate programs at the proper time and executing the steps sequentially. However, this approach only works for processes where there are no unexpected events that require replanning, or where human operators can intervene to make the necessary adjustments. As the uncertainty in the environment increases and the demands for agility grow, plans need to be computed nearer to the time when they will be executed, and be recomputed as execution proceeds, to address unexpected events and to take advantage of unpredicted opportunities. For military operations that depend on the state of many factors that cannot be known ahead of time, operational plans need to be computed close to execution time, and recomputed frequently during the course of operations. The repetition rate of replanning must be at least such that a new plan is generated before the old plan is completed. However, in highly uncertain environments, it is best if the planner generates a new plan before the EXecutor completes the first step or two of the old plan.

The 4D/RCS is designed to support real-time planning. Planning within the 4D/RCS architecture is distributed at many hierarchical levels with different planning horizons at different levels. This facilitates the ability to replan fast enough to keep up with the demands of a rapidly changing environment. At each hierarchical level, the planning functions compute plans that extend from the anticipated starting state out to a planning horizon that is characteristic of that level. On average, planning horizons shrink by about an order of magnitude at each lower level. However, the number of subtasks to move from the starting state to the planning horizon remains relatively constant (by rule of thumb, on average ten). This causes the temporal resolution of subtasks to increase about an order of magnitude at each lower level. Planning horizons at high levels may span weeks, months, or even years, while planning horizons at the servo level span only a few milliseconds.

The number of levels required in a 4D/RCS hierarchy is approximately equal to the logarithm of the ratio between the planning horizons at the highest and lowest levels.

$$N = \log_k (P_H/P_L)$$

where:

N = Number of levels required in the 4D/RCS hierarchy
P_H = Planning horizon at highest level in hierarchy
P_L = Planning horizon at lowest level in hierarchy
k = typical number of steps in plans at each level
$\log_k$ = logarithm to the base k

4D/RCS can accommodate either precomputed plans, or planning operations that recompute plans on demand or on repetitive cyclical intervals. For example, a string of GPS coordinates can be treated as a precomputed path plan, and called when appropriate. On the other hand, path plans can be generated using maps and intelligence reports about enemy positions. These path plans can be recomputed as intelligence reports or observations of enemy movements become available. The 4D/RCS generic planning capability can also accommodate partially completed plans. For example, route plans or movement corridors computed in advance

can be used by 4D/RCS real-time planners to provide constraints for real-time generation of obstacle avoidance maneuvers and detour routes.

4.1.3 Plan Execution

Plan execution is a process where deliberative planning merges with reactive behavior. In plan execution, plans are modified by feedback that provides information about how closely the plan is being followed. Planning is a process that looks into the future and anticipates what needs to be done in the interval between the starting state and the planning horizon. Plan execution is a process that acts in the present to carry out the plan and correct for errors. For each agent(j) in the BG process, there is a EXecutor EX(j) that executes and monitors feedback relevant to the plan for that agent.

The generic structure of an EXecutor is shown in Figure 14. Each EXecutor contains a plan buffer that is loaded with a plan by the plan selector. The plan consists of two parts: (1) a trajectory of planned actions interleaved with (2) a corresponding trajectory of planned states (or subgoals).

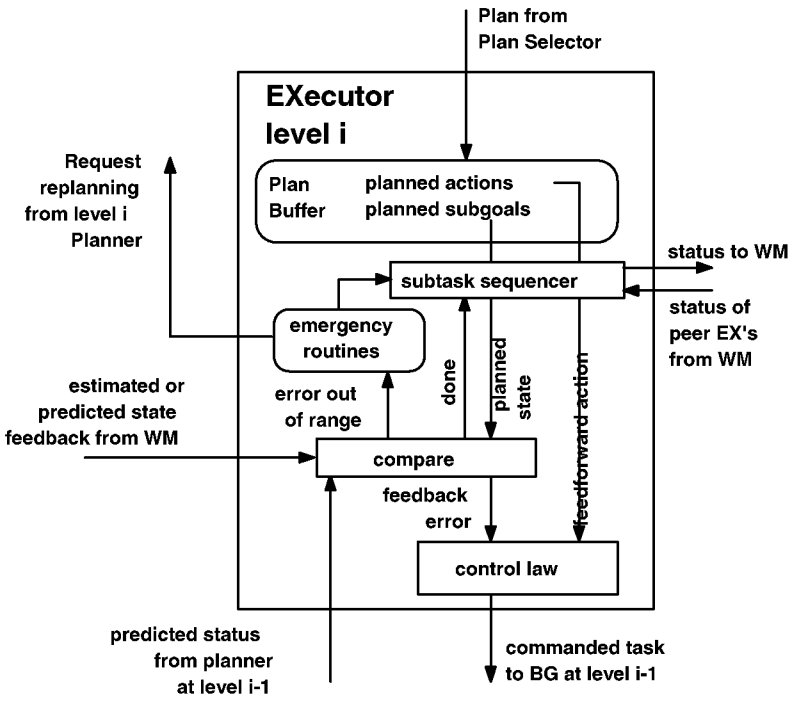


Figure 14. The inner structure of the Executor (EX) subprocess.

In addition to the plan, each EXecutor also receives input in the form of estimated (or predicted) state feedback via Sensory Processing and World Modeling processes from sensors that monitor the state of the world and the internal status of the system, and predicted status from the task planner at level i-1. Status to and from peer EX subprocesses via the WM provide the

basis for coordination between EX subprocesses. Each EX subprocess has a compare operation that computes differences between estimated (or predicted) state feedback, predicted status, and the current planned subgoal. For discrete event systems, status may simply report "busy" or "done" from lower level EX subprocesses.

Df. An **estimated state** is the *WM's best estimate of the state of the world based on all previous sensory input.*

Df. A **predicted state** is a *prediction of the state of the world made by a dynamic model in the WM based on the estimated state and commanded actions.*

Df. A **predicted status** *is a prediction of the status of the system based on the planning process in the lower level BG process*

Typically, feedback controllers use the predicted state if it is available from the WM. For continuous control systems, the estimated or predicted state feedback is compared with the planned subgoal, and a feedback error signal is generated. For a system with a lower level planner, the predicted status of the system can be compared with the planned subgoal, and a predicted status error can be generated. Either type of error can fall in three ranges:

1. Subgoal-done. If the error falls within an acceptably small neighborhood (i.e., tolerance) of zero, the subgoal is accomplished and the compare operation emits a <done> signal. The subtask sequencer outputs the next planned action as a new feed forward action to the control law and sends the next planned state to the compare operation.

2. Feedback compensation. If the error falls between the subgoal-done neighborhood and out-of-range conditions, the feedback error is sent to the control law for error compensation. This enables the EXecutor to function as part of a servo control loop, computing an output designed to null the difference between the estimated state reported by the world model (and/or the lower level planner) and the planned subgoal state.

3. Out-of range emergency. If the feedback error falls in the out-of-range conditions, the emergency action generator is triggered to substitute an appropriate emergency action in place of the current plan. The emergency action provides an immediate response while the planner generates a new plan.

Under normal conditions, EXecutors spend the majority of their time in the feedback compensation mode. The subtask-done mode is a transient condition that triggers a transition from one node to the next in the plan state graph. The emergency error mode should occur only rarely. A typical EXecutor is therefore usually in the process of outputting a feed forward action and computing error compensation to null the difference between a planned subgoal and the estimated or predicted state of the system.

At all but the lowest level, outputs from an EXecutor become an input task command to a Job Assignor in a subordinate BG process at the next lower level. At the lowest level, outputs from EXecutors go to the actuator drive mechanisms.

4.1.4 Feedback and Feed forward

Feedback compensation operates on the error between desired and observed results. Therefore, feedback compensation, by its nature, is always less than optimum. In most cases, feedback errors can be reduced by increasing the servo loop gain. However, there are limits beyond which increasing the gain produces instability. High gain feedback compensation is stable only in systems where loop delays are short, relative to the dynamic response of the system. When delays are significant, feedback compensation may arrive too late to reduce errors, and may produce oscillations causing system instability.

Therefore, for systems with inherent delays (such as the delay between turning the steering wheel and a change in vehicle motion direction), feed forward actions are desirable, and may be necessary, to achieve high performance. Feed forward actions are derived from planning models that enable the controller to anticipate the response of the system to planned actions. If the models are precise and there are no perturbations or noise in the world, then feed forward actions will produce the desired results with no error.

Of course, models are seldom perfect and there is almost always noise and perturbations. Thus, it is rarely possible to precisely model and anticipate everything about the world. There are almost always errors between planned and observed results, and thus there typically is a need for feedback compensation. The best control strategy is to combine feedback compensation with feed forward actions as is shown in Figure 14.

Planned actions can be used as feed forward actions that anticipate the behavior of the system being controlled. Planned states then become desired states, that can be compared with estimated or predicted states, to generate feedback compensation. The EX subprocesses then combines feed forward actions with feedback compensation to provide the best possible control of behavior.

4.1.5 Emergency Actions

In 4D/RCS, feedback errors are detected and addressed at the lowest hierarchical level possible, where the response can be the quickest. At each level, the EX subprocess provides the first line of defense against failure. The EX acts immediately and reflexively, as soon as an error condition is detected, to make a correction by feedback compensation (if possible), or by a preplanned emergency remedial action (if necessary). If either of these two processes is successful, the error is corrected at the lowest level by the highest speed feedback loop, and the agent proceeds with its current plan without intervention by higher levels.

If, however, both error compensation and preplanned remedial actions are unsuccessful, then replanning is required. In this case, the emergency action generator buys time for its planner to replan by executing a preplanned emergency procedure (such as stop, or take cover) until its planner can generate a new plan.

Only when error compensation, emergency action, and replanning at level are all unsuccessful in correcting or preventing failure, will higher level executors and planners become engaged to change tactics or strategy. Thus, error correction occurs at the lowest levels first. Uncorrected errors ripple up the hierarchy, from executor to planner at each level and from lower to higher level BG processes, until either a solution is found, or the entire system is unable to cope, and a mission failure occurs. At any level, the system may request assistance from a human operator or from another system.

4.1.6 An Example Planning-Execution Hierarchy

In Figure 15, a driving task <Go_to_point> requiring about two hours is input as a command to the Platoon level. The planner at the Platoon level looks ahead to a planning horizon about two hours in the future, and decomposes the task into a plan <Go_along_path> from a starting point (S) to a goal point which is the first step in the plan at the next higher level. The path at level 6 is defined by a set of way points, or subgoals (a61), (a62), (a63) . . . (a69) leading to the goal point (a71). If the waypoints are equally spaced, they will be about ten minutes apart.

Figure 15 shows the planning and execution activity at six levels of the 4D/RCS hierarchy as this driving task is carried out. (In this example, the JA aspect of planning is omitted for simplicity. Only SC functions are illustrated.) At each level (i), the planner looks ahead to a planning horizon a time T(i) in the future. It computes a plan consisting of about ten waypoints from the current state to that planning horizon. Thus the plan generated at each level (i) will have steps approximately T(i)/10 apart in time. This yields discrete waypoints on the ground in front of the vehicle with spacing proportional to the vehicle velocity. In the example of Figure 15, a vehicle velocity of 10 meters per second (36 kilometers per hour) is assumed.

Each of the waypoints in the plan at each level will (with modifications by the executor) become a task goal for the BG planner at the next lower level (i-1). Thus, at each lower level of the 4D/RCS hierarchy, the planning horizon shrinks by about a factor of ten, and the distance between waypoints in the plan decreases by a factor of about ten.

In this simple example, a plan is expressed as a simple linear string of waypoints such as might be encountered while driving on an open road. More complex plans might be expressed as a state graph with a number of conditional branch points, such as might be encountered when driving through a network of city streets. For cross-country driving, it may be useful to assign a tolerance that defines a corridor within which the vehicle is free to maneuver.

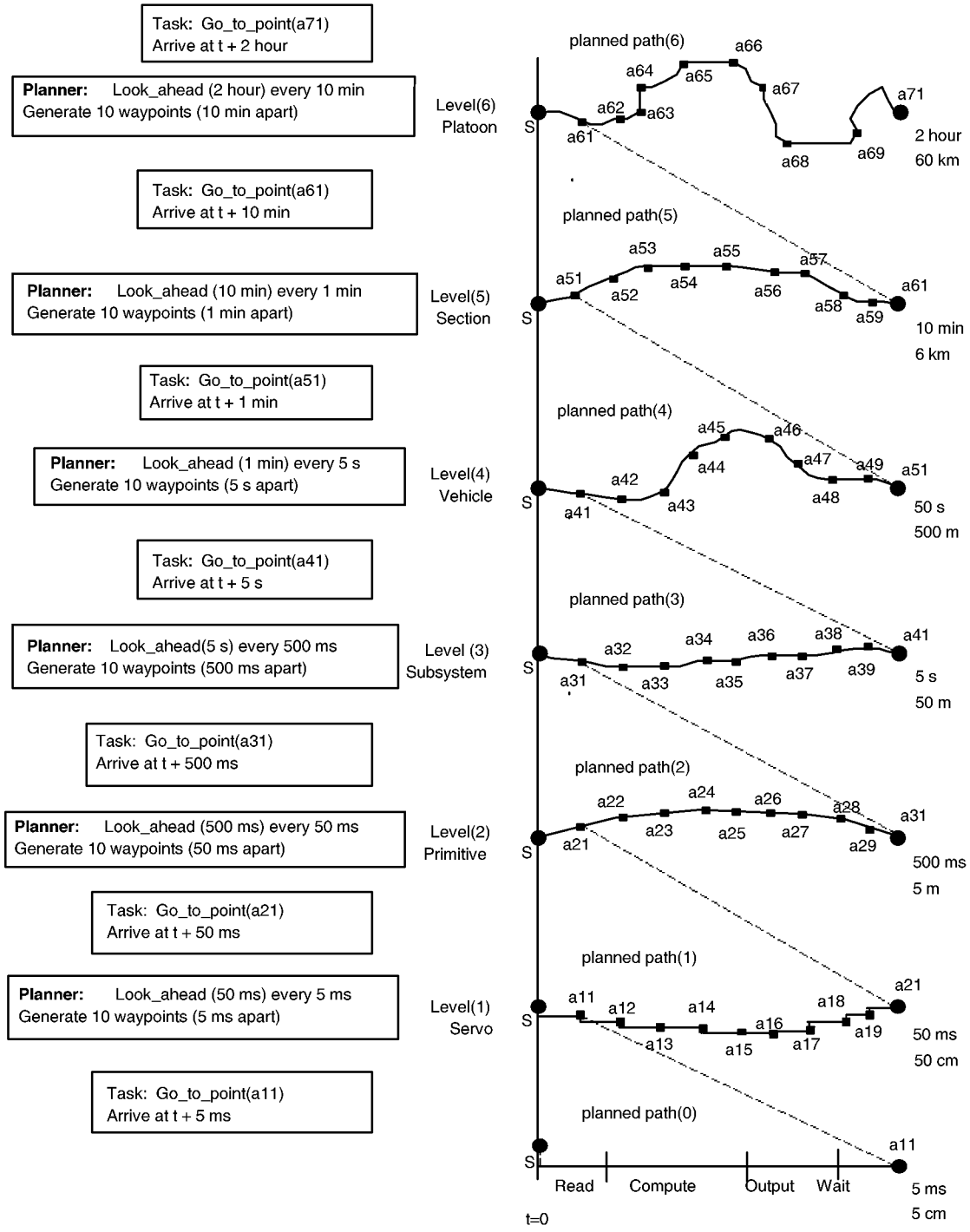


Figure 15. A 4D/RCS timing diagram. On the right, a typical planning horizon and a planned path defined by about ten waypoints is shown for each level. At the bottom, a single servo executor cycle (read, compute, output, wait) is shown. On the left, an example of a typical task command frame and the functions performed by the planner is shown for each level.

As the vehicle moves along its planned path, the planning horizon also moves ahead. For example in Figure 16, when the vehicle is at the start (S), the planning horizon at level (4) looks forward 50 seconds (a distance of 500 meters at a speed of 10 meters per second) to the next goal

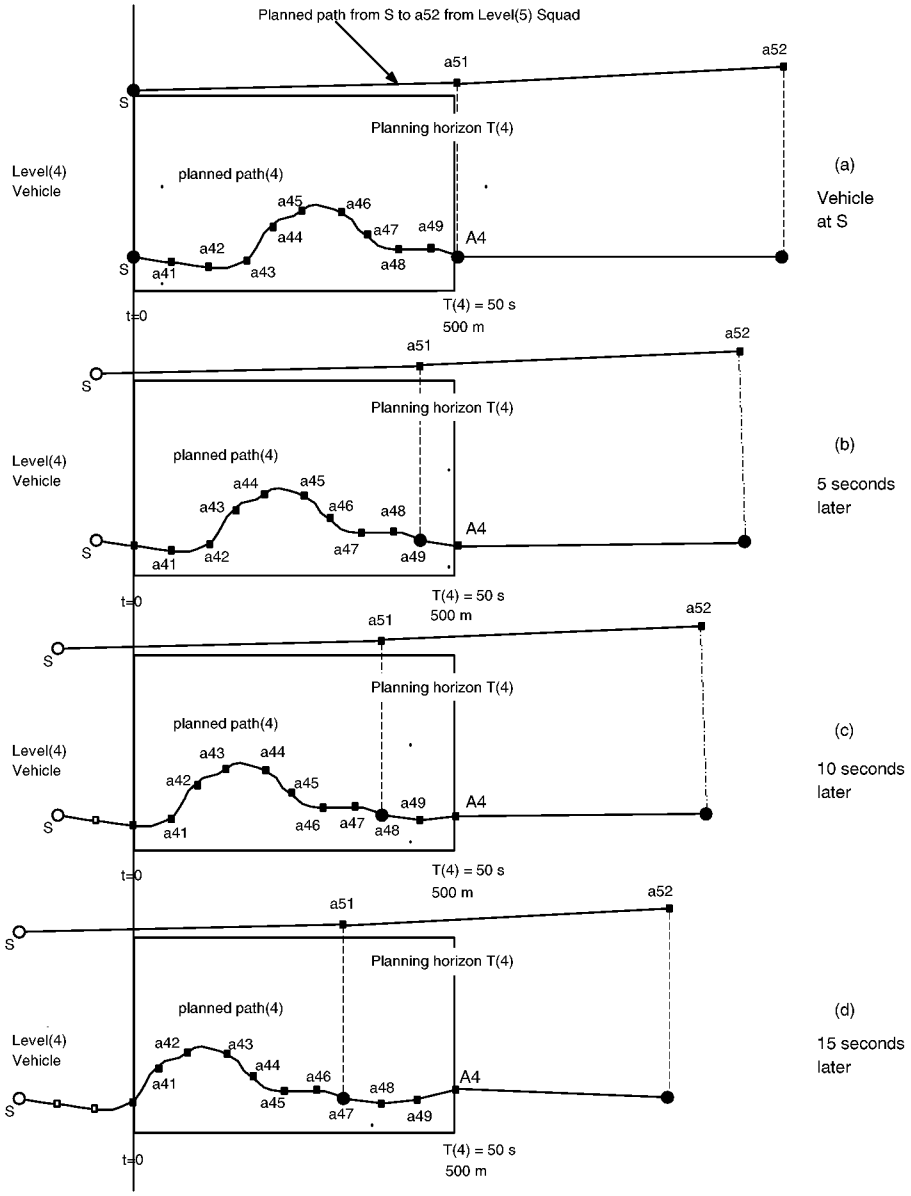


Figure 16. Replanning at T(i)/10 intervals. At time (a), when the vehicle is at point S, input to level(4) is a command <Go-to a51> with a low resolution Section plan to go from S through waypoint a51 to a52. At time (a), the level(4) Vehicle planner generates a higher resolution Vehicle plan (a41, a42, a43, . . . , a49) from S to a planning horizon 50 seconds in the future. 5 seconds later, at time (b), about when the vehicle has reached waypoint a41, the level(4) planner has generated a new plan. The waypoints are shifted down in the plan, and a new waypoint is added at the end. This replanning process is repeated again every 5 seconds, at time (c) and (d).

point from level (5) which is (a51). It generates a planned path (4) consisting of (a41), (a42), (a43) ... (a49) and ending in (a51). As the vehicle begins to move, about 5 seconds later it reaches (a41). The planning horizon is now 50 meters beyond the first goal point (a51). The planner must now begin planning for reaching the second goal point (a52). The planning horizon is indicated in Figure 16 by a rectangular box whose left side is at the present (t = 0), and whose right side is at the planning horizon (t = T (4)).

As can be seen in Figure 16, in order for a planner at level (4) to plan to a moving horizon T(4) in the future, it typically must have access to at least the next waypoint beyond T(4) in the plan of the higher level(5). In Figure 16, the first waypoint in the Section level(5) plan is (a51). The next waypoint beyond T(4) is the second waypoint in the Section level(5) plan which is (a52). Typically, the planner at a level(i) does not require access to more than the next two waypoints at level(i+1).

4.1.7 Planner Timing

For real-time planning, the planner at a level must be able to generate a new plan in less time than it takes to execute the old plan (i.e. <T(i)). Otherwise the system must periodically stop and wait for the planner to complete its work. This type of behavior was typical of early efforts in real-time planning. However, pausing periodically to wait for a planner is unacceptable for Demo III applications.

In dynamic unpredictable environments, such as the battlefield where the world and the goal may be changing rapidly, the planner must be able to replan much faster than real-time. Ideally, the planner should be able to replan in the time it takes to execute the first waypoint in the old plan. In this case, the planner will run every T(i)/10, and must run at 10x real-time. This is illustrated in Figure 16 by a series of four replanning cycles. The planned path from the next higher level is shown as input to the level (4) Vehicle task planner. This input plan consists of waypoints (a51) and (a52). The level (4) Vehicle planner takes this input and generates a planned path(4) that extends out to the level (4) planning horizon T (4) = 50 s. When the vehicle is at position S, a level (4) Vehicle planned path is generated from S to a51. This consists of waypoints (a41), (a42) ... (a49). By the time the vehicle has reached the first waypoint (a41), another plan is generated again out to the planning horizon T (4) = 50 s, which now is beyond the point (a51). All the planned waypoints are shifted left (a41 <- a42, a42 <- a43 ... a48<- a49) and a new way point is generated for (a49). As this procedure is repeated, the vehicle moves along the planned path. As it passes each waypoint on its planned path, it shifts its planned waypoints left in the planned list, and adds a new waypoint at the planning horizon. At each level (i), the planner thus always has a plan that extends out to its planning horizon at T(i). As it achieves each step in its current plan, it computes a new plan with an additional step at the planning horizon. In an uncertain environment, conditions may change as the vehicle moves along its planned path. This may require that the replanning process alter some of the intermediate waypoints in addition to adding a new waypoint at the planning horizon.

It may be possible to conserve computational resources if the planner generates plans with logarithmic resolution – with the first steps in the plan at intervals of T(i)/10, and steps near

the planning horizon at longer intervals. Replanning less frequently may also be used to conserve computational resources. For example, if the replanning interval is T(i)/3, a new plan will be generated only every three waypoints. The disadvantage with this second alternative is that the system will not be able to respond as quickly to changes in the environment. In general, the faster the replanning, the more adaptable is the behavior.

There is a trade-off between the respective capabilities of the planner and executor. The executor has the advantage that it can adapt to changes in the environment within a single control cycle. If the executor can adapt to a wide range of possible conditions, the need for frequent replanning is reduced. On the other hand, the planner has the advantage that it can accommodate a wider range of environmental changes and can exhibit a greater degree of creativity than the executor. If the planner is sufficiently fast, the need for executor adaptability is reduced.

4.1.8 Executor Timing

The timing and frequency of change in the executor output depends on a number of factors. There are four types of conditions that may require the executor to change its output. These are:

- **A task assigned to a lower level BG process has been completed.** This is typically detected by the compare operation that observes that the feedback state is within the pre-determined tolerance of the planned state. This generates a <done> report to the subtask sequencer. The action is to send the next feed forward action to the lower level BG process as a commanded task. For discrete event system, the lower level EX subprocess may report a <done> status which is simply forwarded to the subtask sequencer.

- **The executor has encountered an emergency condition.** This is generated by an <out of range> flag generated by the compare operation. The action required is to output a reflexive response to the emergency condition.

- **A new plan is inserted into the plan buffer by the plan selector.** The action required is to issue a new feed forward action as the first step in the new plan.

- **The current estimated or predicted state of the world varies so that the feedback error changes significantly.** The action required is for the control law to compute a new value of error compensation.

All of these conditions may change at any time, and hence require continuous monitoring by the executor. Therefore, the executor must sample its inputs at a rate fast enough to enable it to respond to sensory feedback in a timely manner. At each level of the sensory processing hierarchy, filter time constants in the sensory processing hierarchy restrict the rate of change in estimated and predicted state variables and hence the effective bandwidth of the servo loop of which the executor is a part. At the lowest level, the time constants are short and loop bandwidth is high. Thus, the lowest level executors need to cycle at the highest rates. In the 4D/RCS

architecture, the servo level bandwidth required depends on the dynamic response of the actuators being controlled. In the mobility subsystem, steering and braking actuators require executor computation cycles on the order of 5 milliseconds. In the attention subsystem, camera pan/tilt servos require similar executor computation cycles. At the primitive level, accelerometers may produce signal bandwidths that also require 5 millisecond execution rates. Data from other sensor systems such as CCD or FLIR cameras produce images that change 30 times per second. LADAR images arrive more slowly but LADAR may be merged with CCD or FLIR data 30 times per second. Thus, executors at the subsystem level may need to compute every 33 milliseconds. Higher level executor computation cycles can be less frequent because estimated and predicted state variables at the higher levels require temporal integration of lower level signals.

4.2 World Modeling

As defined in section 3, the reference model architecture, the world modeling (WM) process is a functional process that constructs, maintains, and uses a world model knowledge database in support of behavior generation and sensory processing processes. The WM process performs four basic functions illustrated in Figure 17:

1) **Maintenance and updating of information** in the Knowledge Database (KD)

- In each node the WM process maintains the KD, keeping it current and consistent.
- The WM process updates the state estimates in the KD based on correlations and variance between world model predictions and sensory observations at each node. This may be part of a recursive estimation loop that operates at the data sampling rate.
- The WM process updates both images and symbolic frame representations and performs transformations from image to symbolic representations, and vice versa.
- The WM process enters new entities into the KD, and deletes entities that are observed to no longer exist in the world.
- The WM process maintains the pointers between KD data structures that define semantic and pragmatic relationships between entities, events, images, and maps.

2) **Prediction**

- The WM process generates short term predictions of expected sensory observations that enable sensory processing (SP) processes to perform correlation and predictive filtering.
- The WM process uses symbolic representations, iconic images, masks, and windows to support visualization, attention, and model matching. Examples of predicted images at lower levels might be what a particular camera would be expected to produce, or what a series of tactile sensor readings might measure. At higher levels, predicted images might be digital terrain maps showing obstacles, buildings, roads, bridges, woods, and other terrain features. At still higher levels, predicted images might be maps of large regions.

3) **Response to queries** for information required by other processes

- The WM process responds to "What Is?" queries from the BG (PL and EX processes) regarding the state of the world or the state of the controller.
- The WM process may provide inferencing capabilities to compute responses for information that is not explicitly stored in the KD.
- The WM process transforms information into the coordinate system requested by the query.

4) **Simulation**

- The WM process performs simulations in response to "What If?" queries in order to support the planning functions of the BG processes.
- The WM process uses structural and dynamic models and rules of physics and logic to simulate the results of current and hypothesized future actions.
- The WM process may use inverse models to compute the actions required to produce desired results.

Examples of simulation at lower levels might be of steering trajectories for high speed cornering. At higher levels, path plans for obstacle avoidance might be simulated. At still higher levels, various routes from one location to another or various formations for tactical maneuvers might be simulated.

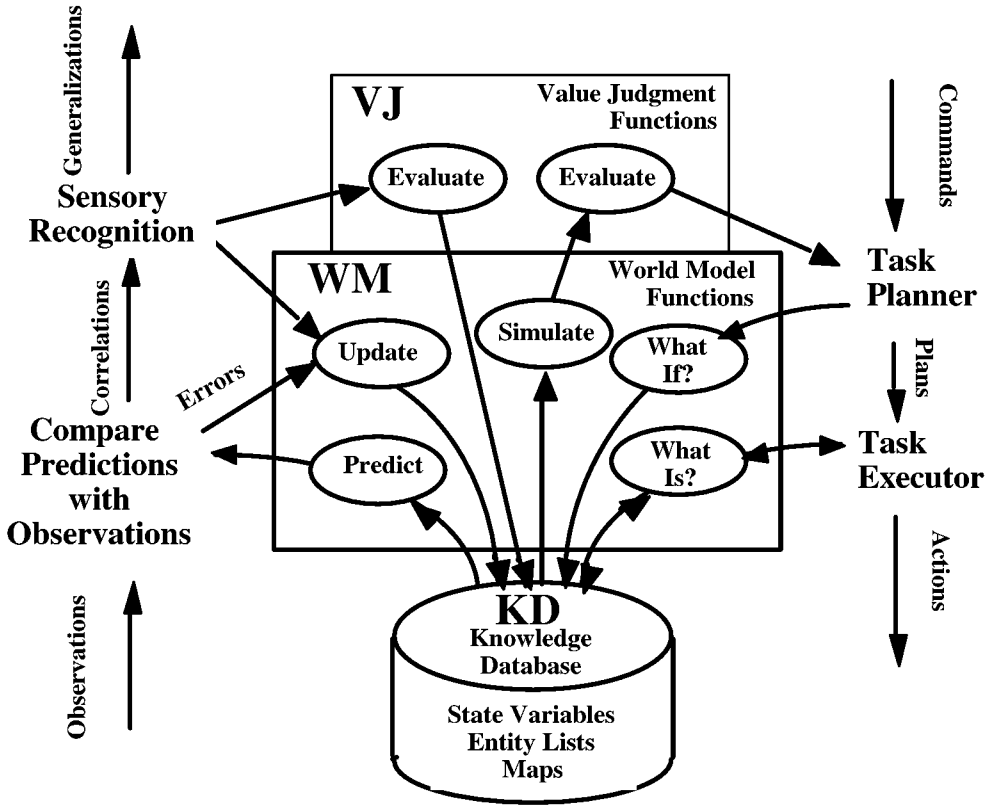


Figure 17. World Modeling (WM) and Value Judgment (VJ) processes. WM and VJ processes typically exist in every node of the 4D/RCS architecture.

4.3 Value Judgment

As defined in section 3, the Value Judgment (VJ) process is a functional process that computes value, estimates importance, assesses reliability, and generates reward and punishment. Figure 17 also shows a block diagram of the functional operations and data flow pathways for interactions between world modeling and value judgment. VJ processes evaluate plan results simulated by the WM processes. VJ processes contain algorithms for the following:

- **Computing** the cost, risk, and benefit of actions and plans
- **Estimating** the importance and value of objects, events, and situations
- **Assessing** the reliability of information
- **Calculating** the rewarding or punishing effects of perceived states and events.

VJ processes compute the cost functions that enable intelligent behavioral choices. VJ processes define priorities, set limits on risk, and decide how aggressive or conservative a system should be in pursuing its behavioral goals. VJ processes assign values to objects, events, and

situations recognized by SP processes. These assigned values may be used to compute whether it is worthwhile to defend a position or to attack a target. Values placed on "friend" objects may define how valuable they are, how vulnerable to attack, and how much in need of defense or rescue. Values placed on "foe" objects may define how dangerous they are or how aggressive they are likely to be toward the self-vehicle. Values placed on regions of space may define how safe or risky it is to occupy that space, how costly to traverse it, how valuable a region of space is, how worth defending, or how valuable to occupy.

Values placed on objects or regions of space can be overlaid on images or maps. This enables image processing techniques to be applied to cost values as they are to other attribute values such as intensity, color, or range. Values of cost and risk in an image can be analyzed to find the lowest cost regions for path planning. Gradients of cost and risk can be analyzed to generate commands for steering, or aiming. VJ processes may compute statistics on the reliability of information about the world based on the correlation and variance between observations and predictions. Confidence values can be assigned to state variables and regions of the visual field. VJ processes evaluate perceived and planned situations to enable BG processes to select goals and set priorities for behaviors. VJ processes also compute what is important (for attention) and what is rewarding or punishing (for learning).

Cost values can also be associated with edges in a planning graph. A planning graph may be a more complex structure than can be represented in a 2-dimensional array or image. And there may be more than a single cost value associated with an edge in a planning graph. For example, the cost of traversal between two nodes in a graph may be direction-dependent (e.g., a one-way street, or the slope of a hill.)

VJ processes compute different types of values at different hierarchical levels. For example, at levels one and two, VJ may compute of cost functions for path planning that minimize jerk or acceleration. At levels three and four, VJ may minimize energy or time. At levels five and six, VJ may minimize risk or exposure to enemy action.

4.4 Knowledge Database

The Knowledge Database consists of data structures and the static and dynamic information that collectively form a model of the world. The KD is the information needed by the WM to support the BG, SP, and VJ processes in each node. Knowledge in the knowledge database includes the system's best estimate of the current state of the world plus parameters that define how the world state can be expected to evolve in the future under a variety of circumstances.

Types of data structures include scalars, vectors, arrays, symbols, strings, pointers, lists, frames, and graphs. Information in the knowledge database include signals, state variables, names, characters, numbers, attribute values, relationships, images, maps, rules, equations, and recipes. Knowledge includes structural and dynamic models that describe how the world behaves, and task knowledge that describes how to do tasks, what tools to use, what resources are needed, and what information is required. State variables define estimated conditions in the

world. Attributes describe properties. Entities and events can be represented in frames that contain lists of state variables, attribute values, and relationships. Images contain information about the position of entities in the world, and maps can provide geometric representations of terrain overlaid with labels, icons, and text that contain information necessary for situation assessment and planning of action.

4.4.1 Signals

Df. A **signal** is *output from a sensor that measures a phenomenon in the environment.*

A signal may represent an attribute such as amplitude, intensity, frequency, time delay, temperature, color, position, velocity, or force. An attribute can be represented by a time dependent scalar of a vector variable. For each attribute variable, temporal derivatives and integrals can be computed to several orders. Signals may be filtered to remove noise or detect frequencies or rates of change.

4.4.2 Entities

Df. An **entity** is *something existent with continuity of geometric structure.*

It is important to distinguish between entities that exist physically in the world and entities that exist either mentally in the mind or logically in an intelligent system's knowledge base. We will designate entities in the world as external entities and entities in the mindor in the intelligent systems as internal entities.

Df. An **external entity** is an *entity that exists in the real world external to the intelligent system.*

External entities typically are clumps of matter that occupy space and are physically connected in some way. For example, an external entity may be an object such as a rock or a post. It might be a fireplug, an automobile, a person, a hat, a chair, a building, a window, a stairway, or a step. It might be a surface, such as that of a leaf, a meadow, or a road. It may be an intersection of two roads. It may be the edge of a building or the point of a needle. It may be the end of a line, or a dot on a page. An external entity might be a bush or tree consisting of hundreds or thousands of leaves, branches, and twigs. It might be a vine that may climb on and intertwine with the branches of a tree or bush. It might be a river with many tributaries that sometimes flood - and sometimes dry up. It might be a lake with miles of coast-line consisting of rocks and inlets with waves that surge and ebb. External entities may also be groups of things that are not physically connected, but possess common attributes such as spatial proximity or similar motion. For example, an external entity may be a cloud of smoke, a swirl of dust, a flock of birds, a school of fish, a wave of water, a crowd of people, a stream of traffic, a forest (consisting of hundreds of trees and bushes), falling rain or snow, or a lawn (consisting of millions of blades of grass).

External entities often correspond to things in the external world that can be measured or counted, such as pebbles, grains of sand, drops of water, items of clothing, furniture, structures, or vehicles. Manufactured entities typically are well defined with surface properties that can be precisely measured. Edges may be defined in terms of intersecting surfaces, or by sharp radii of curvature in the surface of the object, or by sharp discontinuities in range or color. Many natural entities are not clearly defined and cannot be precisely measured. For example, where exactly does a mountain begin and a valley end? Where is the bottom of a sand dune, or the edge of a weather front? The answers to these questions depend on the scale and resolution of the measurement.

These examples suggest that the concept of an entity is not inherent in the environment, but is a hypothesis imposed on the world by an observer in the intelligent system to facilitate an interpretation of sensory input from the world. External entities are simply the observer's way of organizing data about the world in a manner that increases the probability of successful behavior.

Df. An **internal entity** is *a data structure in the knowledge database that represents a hypothesized external entity*

An internal entity corresponds to a concept. It is an internal representation of something real or imagined. Internal entities are things the intelligent system perceives about the world. They exist only as data structures in the intelligent system. An internal entity may consist of a labeled group of pixels in an image, or a symbolic frame that contains attributes, state, and relationship pointers. An internal entity may represent an external entity such as a rock, or a tree, or the surface of a road or a building. However, the internal entity is only a data structure that contains information about the external entity. It is simply the observing system's best estimate of what exists in the world.

It is possible to have internal entities with no corresponding external entity at all. For example, an intelligent system may mistake a wooden model of a tank for a real tank, or a camouflaged tank for a clump of bushes. Camouflage and deception that create incorrect representations of reality in the mind of the enemy are common tools of warfare.

So how does the intelligent system know whether there is any corresponding external reality? The short answer is: "It doesn't." Any system can only estimate the probability that its internal entities have corresponding external entities. When predictions based on internal hypothesized entities correctly predict the future, the probability that the hypothesized internal entities correspond to external reality is increased. When predictions based on hypothesized internal entities fail to predict the future, the probability that the entity hypothesis is correct is reduced. If the probability of correctness rises above an upper threshold, the system accepts the entity hypothesis as true and acts accordingly. If the probability of correctness falls below a lower threshold, the entity hypothesis is rejected as false and an alternative entity hypothesis is generated.

These examples illustrate only some of the deep philosophical issues surrounding the problem of knowledge representation. Sufficient for the discussion here is to note that all internal representations are hypothetical constructs internal to the intelligent system that enable it

to interpret sensory input. It is only a hypothesis that a corresponding external reality exists. What is important is not whether a hypothesis is true, but whether it has functional utility, i.e., whether is produces a representation of the world that is useful in generating successful behavior.

The utility of a hypothesized entity can be measured by its ability to explain the past and predict the future. The criterion is whether an entity hypothesis produces significant performance benefits. An intelligent machine that can accurately predict even a few seconds into the future is more likely to survive and achieve behavioral goals than one that can only react to events after they occur. A system that is able to predict hours or days into the future has an enormous behavioral advantage. Reliable prediction enables the intelligent system to plan and take preemptive action to avoid danger, maximize benefit and payoff, minimize cost and risk, and win in competition with other systems that have inferior predictive powers. Whether internal entities actually correspond to external reality is largely irrelevant and in many cases may be undecidable. All that really matters is whether hypothesized internal entities increase the reliability of predicting the future and hence the probability of successful behavior.

Df. An **entity attribute** *is a property of an entity.*

Entities have attributes that are properties of the entity as a whole. For example, a pixel entity may have attributes such as intensity, color, spatial or temporal gradients, range, position in the image, and image flow rate and direction. An edge entity may have attributes such as length, curvature, position, velocity, orientation, magnitude, and type (e.g., range-edge, brightness-edge, color-edge, texture-edge, surface-intersection-edge). A surface entity may have attributes such as area, curvature, position, velocity, orientation, temperature, and texture. An object entity may have attributes such as volume, shape, position, velocity, orientation, temperature, and mass. A group entity may have attributes such as volume, density, shape, position of the center of gravity, mean velocity, variance about the mean, and relationships between group members.

Entity attributes may exist in three forms:

1. **Observed attributes** that are derived directly from sensors, or computed from other observed attributes, e.g., an observed edge velocity.
2. **Estimated attributes** that are computed from observed attributes by recursive estimation or another filtering technique, e.g., a best estimate of an edge velocity.
3. **Predicted attributes** that are computed from estimated attributes using knowledge of system dynamics, geometry, and control actions, e.g., a prediction of the future value of an edge velocity.

The attributes of an object that vary over time (such as position, velocity, energy, momentum, or temperature) are often called state variables.

Df. State variables are *attributes that define the state of the entity or process.*

A state is typically realized with a data structure containing a set of state variables for representing the dynamic condition of an entity or a process. The state of a physical object

typically refers to its position and velocity (and sometimes higher derivatives) at a point in time. The state of an electric circuit may refer to voltage or current levels at specific points in the circuit. The state of a finite state machine refers to the node in a state graph, or the line in a state table, that is active. The state of a software process may represent the point of execution in a flow chart, the contents of an instruction counter, or the currently executing statement in a program at a point in time. The state of a physical process may represent the temperature, pressure, density, volume, chemical composition, or flow rate of the participants in a reaction at a point in time. The state of a situation may represent the relationships that exist between entities in an environment at a point in time. The state of a control system normally refers to the current condition of the system. In the ideal case where there exists a perfect system model and complete knowledge of future inputs, knowing the current state enables the prediction of all future states. An entity or process can have past, present, or predicted future states.

The definition of state is context sensitive. An entire control system may have a state. Each individual node in a 4D/RCS control system may have its state. A process inside of a node may have a state.

An entity, together with its attributes, state, and relationships to other entities may be represented in the knowledge database in symbolic form as an entity frame.

Df. An **entity frame** is a *data structure used for representing an entity and contains the entity name, a state, a list of attribute-value pairs, a pointer to a parent entity, pointers to sub-entities, a pointer to the entity image and/or map in which the entity can be observed, and pointers that define relationships with other entities or events.*

An entity frame contains information that the intelligent system knows about an entity. Figure 18 shows an example entity frame. The entity name is an address or index by which the entity frame can be accessed in a database or library of entities, and to which other entity frames can be linked. An uncertainty parameter can be associated with the name to indicate how certain the system is that the entity has been properly identified. The entity frame contains estimated entity attributes that are the system's best estimate of entity color, size, and shape. Entity attributes are characteristics and properties of the region occupied by the entity. Observed entity attributes can be computed from sensor signals by integrating attributes of the pixels belonging to the entity in the image [Adelson and Movshon82]. Predicted entity attributes can be computed from estimated entity attributes by temporal projection through the world model.

Entity state-variables describe dynamic properties such as position, orientation, and velocity of the entity in a particular coordinate system. These can be used as parameters in world modeling processes for prediction and simulation, and by sensory processing functions for classification, detection, and recognition. Uncertainty parameters can be associated with state-variables to indicate the dependability of the estimates of these values. Observed state-variables are typically defined in sensor egosphere coordinate system. Estimated and predicted entity state-variables may also be in egosphere coordinates, or may be transformed into some other more convenient coordinate system.

```
NAME = entity_id (uncertainty)          // this is the frame address in the KD

// attributes – these are characteristics that address the question  What?
estimated color              = red, green, blue intensities
estimated size               = length, height, width dimensions
estimated shape              = curvature, moments, axes of symmetry, etc.

//  state – these are dynamic properties that address the question  Where?
estimated position           = azimuth, elevation, range, (uncertainties)
estimated orientation        = roll, pitch, yaw (uncertainties)
estimated velocity           =v-azimuth, v-elevation, v-range, v-roll, v-pitch, v-yaw (uncertainties)

// class – pointers to the entity image and classes to which the entity belongs
entity image        = pointer to the entity image in which the entity appears
generic class1      = pointer to the generic class1 prototype
generic class2      = pointer to the generic class2 prototype
generic class3      = pointer to the generic class3 prototype

// value or worth of the entity
worth to preserve   = value of preserving
worth to acquire    = value of acquiring
worth to defend     = value of defending
worth to defeat     = value of defeating

// pointers that define parent-child relationships
belongs to          = pointer to parent entity
has part1           = pointer to subentity1
has part2           = pointer to subentity2
has part3           = pointer to subentity3

// pointer that define situational relationships
on top of           = pointer to entity below
beside-right        =pointer to entity on right

// queues that store short-term state history and expected future states
short term memory           = pointer to STM queue
short term expectations     = pointer to STE queue

// functions that define behavior
behavior1           = responds-to
behavior2           = acts-like
```

Figure 18. The structure of a typical entity frame. Each entity frame consists of a name, a list of estimated attributes, and a set of pointers to other entities. The list of pointers includes a pointer to the entity image that contains regions labeled with the entity_id.

Entity state-variables are time dependent. Thus, there exists a string of states that define a trajectory through state space. This can be represented in the entity frame by a queue of states that store a short-term memory trace for each entity. Predicted future states of each entity can also be represented in an entity frame by a queue of states that represent short-term expectations for each entity.

An entity frame may contain pointers to the entity image in which the entity appears. For example, an object entity will have a pointer to the object image in which it appears. This can enable a graphic engine to overlay entity names or attributes on the image or map. An entity frame may also contain pointers to the class or classes to which the entity belongs. For example an object entity frame may have a generic class pointer that identifies it as a member of the class

of trees. The same object entity frame may have a second generic class pointer that identifies it as a member of the class of pine trees, as well as a specific class pointer that identifies it as a particular pine tree.

The entity frame may contain value attributes that define how valuable an object is as a target (if a foe), or how worth defending (if friend). Value attributes may also define how much a particular entity or situation should be feared or avoided, what a particular location may be worth as a vantage point or as a source of shelter or food. This enables VJ to compute whether or not it would be worth the cost and risk of acquiring it or defending it.

The entity frame may contain pointers that define inheritance relationships with other entities. Each entity frame has a pointer to the frame of a parent entity of which it is a part, and a set of pointers to frames of sub-entities that are its parts. For example, an object entity frame typically will have a parent pointer to the entity frame of the group to which the object belongs. It will have a number of sub-entity pointers to the surface entity frames of the surfaces and boundaries that are its parts. Similarly, each surface entity frame will have a parent pointer to the object entity to which it belongs, and a set of sub-entity pointers to the list entities that are its parts. Each list entity frame will have a parent pointer to the surface entity frame to which it belongs, and a set of sub-entity pointers to the pixels that are its parts. Inheritance pointers are established and maintained by grouping and classification operations performed by sensory processing functions at various levels in the hierarchy.

The entity frame may also contain pointers that define spatial, temporal, causal, or other types of relationships that pertain to that particular entity. An entity frame may include a set of functions that describe the behavior of the entity under certain conditions or in response to certain stimuli. Simple functions may define the behavior of objects under the influence of gravity or friction. Complex functions may define how the entity might be expected to respond to an attack or a gesture of friendship. Behavioral functions may include parameters such as speed, endurance, strength, or range of weapons. Behavioral functions and parameters may be inherited from generic or specific class prototypes.

A block diagram of an entity frame is shown in Figure 19. Entity frames may stand alone as data structures, or be linked to form lists, strings, words, sentences, networks, and maps. Named entity frames provide the basic building blocks of language – nouns. Named entities may be used as the subjects or objects of sentences, and the agents or objects of action. In language applications, entity attributes may serve as adjectives that describe characteristics of the entities to which they are attached. Entity frames can easily be implemented as objects or classes in modern computer languages such as Ada, C++, or Java.

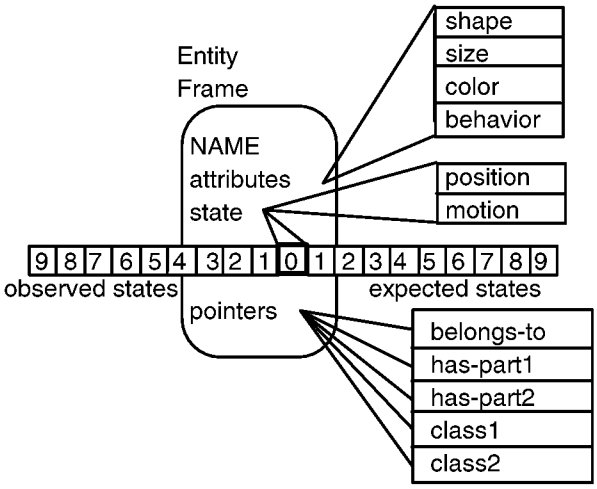


Figure 19. An entity frame consisting of a name, an attribute vector, a state vector, a queue of state vectors consisting of observed and expected states, and a set of pointers that describe relationships with other patterns.

Entity frames can be interconnected by pointers to form causal, semantic, and situational networks. Situational networks may represent situations or geometric relationships such as *"on-top-of", "beneath", "to-the-right-of", "in-front-of", and "inside-of"*. Situational networks may also have pointers to maps or images that pictorially display spatial relationships. Causal networks represent the cause-and-effect relationships between entities, events, situations, and actions that occur in the world. Semantic networks represent the relationships between entities, attributes, situations, actions, and events that define meaning and enable reasoning, logic, and language.

4.4.3 Classes of Entities

There are several stages in which an intelligent system acquires knowledge about external entities in the world. At each stage, it is possible for the intelligent system to take action using the knowledge acquired at that stage. This enables the intelligent system to act quickly and efficiently without waiting for a complete analysis of every entity in every situation. For example, it is sufficient for an insect flying through a complex environment to simply know where surfaces are located relative to its intended flight path. A house-fly does not need to recognize objects to avoid hitting them. All it needs to know is that regions of space are occupied and unavailable for flying.

There are at least three types of internal entity classes that are useful for behavior generation in intelligent systems:

1. **Geometrical** entity classes
2. **Generic** entity classes
3. **Specific** entity classes.

Geometrical Entity Classes

Geometrical entity classes are defined by the geometric properties of entities. Geometrical entity classes include point, list, surface, object, group entity classes. Each of these classes can be represented by geometrical entity frames. These classes are related to each other by a taxonomy of grouping relationships. List entities consist of groups of point entities. Surface entities consist of groups of list entities. Object entities consist of groups of surface entities. Group entities consist of groups of object entities. Each sub-class of geometrical entity class is described in more detail as follows:

Point Entity Classes (Level 1):

Point (or pixel) entity classes have attributes that can be measured by a single sensor at a single point in time and space, or that can be computed at a single point (or over a single pixel) in time and space. Point attributes may describe the properties of a single pixel in an image. Examples of pixel attributes are intensity, color (red, green, blue), range, spatial and temporal gradients of intensity or range, flow direction and magnitude. Point attributes may also describe the output of a single sensor, such as a position, velocity, torque, or temperature sensor, at a point in time

List Entity Classes (Level 2):

List entity classes consist of sets of point (or pixel) entities that satisfy some gestalt grouping hypothesis [We 58] over space and/or time. List entity classes include edge, vertex, and surface patch entities. For example, an edge may consist of a set of contiguous pixels for which the first or second derivatives of intensity and/or range exceed threshold and are similar in direction. A vertex may consist of two or more edges that intersect. A surface patch may consist of a set of contiguous pixels with similar first or second derivative of intensity and/or range and similar image flow vectors. Edge, vertex, and surface patch entity attributes are computed over the set of points that comprise the entity. For example, edge entity attributes may represent the orientation, length, and curvature of the edge, the sharpness or magnitude of the discontinuity at the edge, as well as the centroid of the group of points that make up the edge. Vertex attributes may describe the relationship between the set of edges that make up the vertex. For example, vertex attributes may define the type of vertex (e.g. an endpoint, V, T, or Y), the orientation of the vertex, and perhaps the angles between lines forming the vertex. Surface patch attributes may describe the collective properties of the set of connected points that make up the patch. For example, surface patch attributes may define the position, and velocity of the surface patch, the texture, and the orientation of the surface patch relative to the viewing point.

Surface entity classes (Level 3)

Surface entity classes include surface and boundary entities. Surface entities consist of sets of contiguous list entities that satisfy some surface gestalt grouping hypothesis. For example, a surface may consist of a set of contiguous surface patches that have similar range, orientation, texture, and color. A boundary may consist of a set of edge entities that

are contiguous along their orientation. Surface and boundary attributes are computed over the entire set of points that are included within the entity. Surface attributes may describe the properties of the surface, such as its area, shape, roughness, texture, color, position and velocity of the centroid, etc. Boundary attributes may describe the properties of the boundary between two surfaces computed over the set of points that make up the boundary. For example, boundary attributes may define the shape of the boundary, its orientation, its length, its position, and velocity, which side each surface lies on, etc.

Object entity classes (Level 4)

Object entities consist of sets of contiguous surface and boundary entities that satisfy some object gestalt group hypothesis. For example, an object may consist of a set of surfaces that have roughly the same range and velocity, and are contiguous along their shared boundaries. Object attributes are computed over the entire set of points that are included within the object. Object attributes may describe the properties of an object, such as its volume, shape, projected size in the image, color, texture, position and velocity of centroid, orientation and rotation about the centroid.

Group entity classes (Level 5)

Group entity classes consist of sets of objects that have similar attributes, such as proximity, color, texture, or common motion. Group attributes are computed over the entire set of objects that are included within the group. Group attributes may describe the properties of a group, such as the number of its members, size, density, position, velocity, average direction of motion, and variance from the mean.

The above taxonomy of geometric entity classes was chosen because it parallels common approaches to image processing, and because it appears to bear some resemblance to image processing in the brain. The five levels of geometric entity classes correspond to 5 levels of grouping of lower-level entities into higher-level entities. Pixels sum together, or group, all the incident radiation falling on the region occupied by a photodetector in the image plane of a camera or retina. List entities group pixels into edge fragments and surface patches that can be given a name, and for which attributes can be computed. Surface entities are groups of list entities. Object entities are groups of surface entities. Group entities are groups of objects. This is a useful approach to segmenting an image into regions with attributes that can be measured. For example, in a typical image one can measure the length of an edge, the area of a surface, the projected size of an object, and the density of objects in a group. Entity attributes may include shape, color, temperature, position, orientation, and motion. Geometric attributes can be measured in the 2-dimensional image in terms of azimuth and elevation angles.

If range can also be measured (or estimated), the size, shape, position, orientation, and motion of entities in 3-dimensional space can be computed. This allows entities to be visualized from perspectives other than the camera or eye. For example, entities can be placed on a map. Unambiguous spatial relationships can be determined in the three dimensions. These are required for planning tasks and paths through the environment. The ability of a system to build a geometric model of the space around itself, and to know unambiguously the range as well as

azimuth and elevation of every point in the visual image is the first step in image understanding. Once 3-dimensional geometry of the world is represented and this 3-dimensional space is segmented into surfaces, boundaries, objects, and groups with spatial and temporal attributes (i.e., geometric and dynamical properties), the perceptual process is ready for the next phase of sensory processing and world modeling, namely, classification into generic and specific classes.

Generic Entity Classes

Generic entity classes may include roads, buildings, grass, trees, bushes, water, dirt, sand, sky, and clouds. For any generic entity class, there exists a set of attributes that define a template (or prototype.) The class prototype exemplifies the class and defines the criteria for deciding whether a particular geometric entity is or is not a member of a generic entity class. Observed geometrical entities that have attributes matching those of the generic entity class prototype may be classified as belonging to the generic entity class. The set of attributes that define a class prototype are called criteria for recognition.

Df. An **entity class prototype** is *an entity image or frame that exemplifies or provides the norm for an entity class.*

The entity class prototype is an ideal example of the class. It is the standard by which geometric entities can be assigned to generic classes. An entity class prototype may be represented in the world model knowledge database in either iconic or symbolic form, or both. For example, Figure 20 shows both a prototype image and a prototype frame for a generic object class – an M1 tank. A generic entity class may have many prototype images, taken from different perspectives under a variety of different conditions. An entity class typically has only one, or at most a few, prototype frames. This is because the attributes in the frame representation tend to be invariant with respect to perspective and viewing conditions.

Generic class prototypes may exist for any geometric entity type (pixel, list, surface, object, or group entities). For example, pixels in aerial or satellite photos are often classified as belonging to a generic entity class on the basis of color or multi-spectral image values. A list entity patch may be classified as a road patch if it has the color and texture of a generic road patch template. Edge entities may be classified on the basis of orientation, magnitude, or type (intensity, color, or range edges). A geometric edge entity may be classified as a road edge if it has attributes similar to those of a generic road-edge class template. Surface entities may be classified on the basis of texture, orientation, shape, size, color, or motion. A geometric surface entity may be classified as the side of a building if it has attributes that match those of a generic building-wall class template. Object entities may be classified on the basis of shape, size, material, color, texture, position, orientation, or motion. Group entities may be classified on the basis of shape, size, density, or motion.

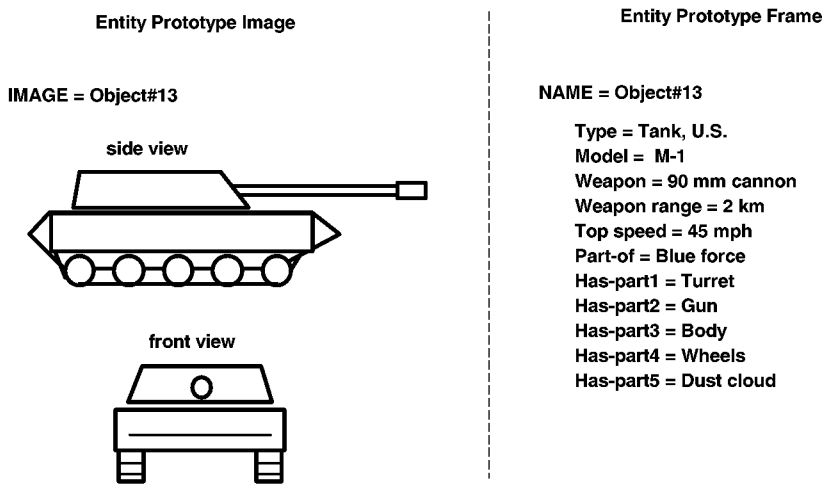


Figure 20. An entity prototype for image and frame representations. The entity prototype image may be a stored photograph, or an image generated by a graphics engine from symbolic data such as a Computer-Aided Design (CAD) data file. The entity prototype frame provides a symbolic description of a typical entity in the generic class. The entity frame may include information that is not available from sensory input. For example, the range of the weapon may be known from the generic class of the tank, but cannot be measured from sensory data.

An intelligent system may have thousands of generic entity classes. The problem with having a large number of classes is not the amount of memory required to store them, but the number of comparisons required to classify geometric entities as members of generic entity classes. As the number of classes grows, number of computations required to make a classification grows, as does the probability of misclassification. The number of comparisons required for classification can be addressed by parallel computational methods. The probability of misclassification can be reduced by including context from higher levels, and constructing lists of entities of attention based on task requirements and priorities. Such lists can be compiled using both top down and bottom up inputs to evaluate importance and likelihood of occurrence in a given task environment.

Specific Entity Classes

A specific entity class has only one member. A specific entity is in a class all to itself. It is a particular instance of a thing. It has unique attributes that distinguish it from all others. A specific object entity may be a specific person, a particular building, or a unique tree. A specific surface entity may be a particular surface of a specific object. A specific list entity may be a particular edge of a specific surface. A specific entity prototype contains the attributes that uniquely identify an entity as the one and only entity in that class. If the attributes of an observed geometric entity match the attributes of the specific entity class prototype, the observed entity is recognized as the specific entity.

There can be a taxonomy of entity classes. For example, a specific tree may be a member of the generic class of oak trees, which is a member of the generic class of deciduous trees,

which is a member of the generic class of trees, which is a member of the generic class of plants, which is a member of the generic class of living things. This taxonomy of classes can be represented in the knowledge database by a *set of pointers*, or links, between classes. A pointer to a more generic entity frame defines an "is_a" or "belongs_to" link. Pointers to more specific classes define "an_example_of" or "contains" links.

4.4.4 Images

Df. An **image** is a *two-dimensional array or manifold of attribute values.*

Images may be generated in a number of ways. For example, an image may be formed by a sketch, drawing, or array of dots on a sheet of paper. Images can be formed by the projection of electrons on the face of a cathode ray tube, or by the optical projection of light from a scene in the world through a lens onto a focal plane or surface. If the focal surface is covered with a photosensitive film, an image will be represented by color or density changes in the film. If the focal surface is covered by an array of photoreceptors such as in a CCD TV camera or the retina of an eye, the image will be represented as electrical charges on the receptor array. Images can also be generated by internal mechanisms (such as a computer graphics engine) from information stored in symbolic entity frames such as vectors or polygons.

An array of photodetectors, on the focal plane of a TV camera, a FLIR, a retina, or the compound eye of an insect, spatially quantizes the image into picture elements. A scanning mirror and pulsed laser beam spatially quantizes a LADAR image. Each photodetector integrates the energy falling on it to produce a signal. The region on the egosphere covered by each photodetector is represented by a pixel.

Df. A **pixel** is a *picture element.*

A **pixel** is the smallest distinguishable region in an image. The region within a pixel has no discernible internal structure.[7] The signal from each pixel in a TV or FLIR image corresponds to an intensity or color attribute integrated over the entire area of the pixel. The signal from each pixel in a LADAR range image may simply represent shortest (or longest) range detected within the pixel, or may list several ranges corresponding to multiple returns from a single laser pulse.

In the RCS Knowledge Database (KD), there can be four types of images: 1) attribute, 2) entity, 3) class, and 4) value images.

Df. An **attribute image** is *a two-dimensional array of attribute values.*

---

[7] This is true for 2-dimensional images. However, a LADAR may produce an image with multiple values of range at each pixel. For example, a single LADAR pixel may contain a first range value from a laser beam striking a wire, and a second range value from the same beam striking a building behind the wire. In fact, a LADAR beam may produce multiple returns when pointed at a bush or stand of tall grass or weeds. Multiple returns can also be produced by falling rain or snow, or clouds of dust or smoke.

In an attribute image, each pixel contains the *value* of the attribute that is measured or computed at that pixel. There are a number of attributes that can be computed at each pixel and hence a number of attribute images. These attributes include: intensity, color (red, blue, green), stereo disparity, range, image flow magnitude and direction, texture, surface orientation, and spatial or temporal gradients of intensity, color, or range (and possibly multiple range images.) For each attribute, an attribute image can be constructed and maintained in registration with the other attribute images. This produces a three dimensional array of pixel attributes as illustrated at the top of Figure 21.

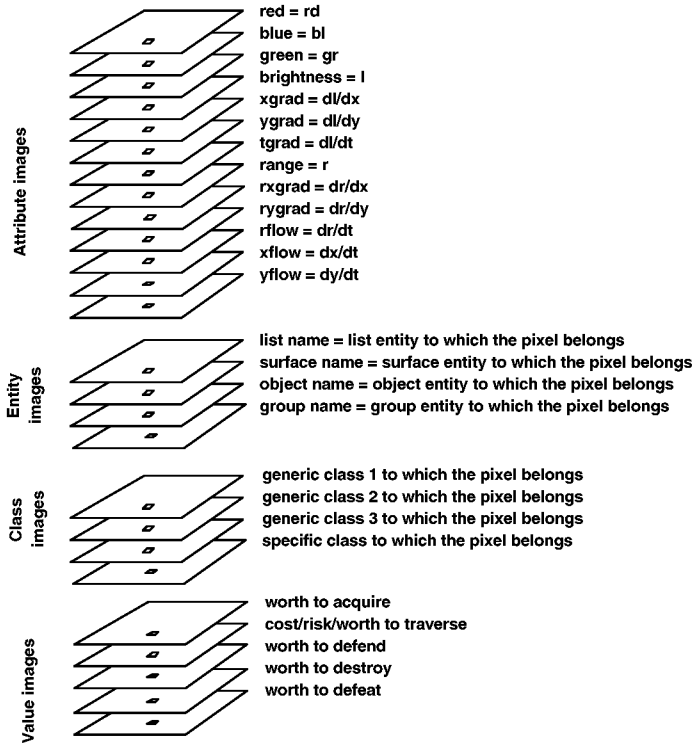


Figure 21. Attribute, entity, class, and value images. The result of image processing is to compute a set of images that are registered with the original retinal image. Value images represents information that is needed for setting priorities and making behavioral plans and decisions.

Attribute images add information to the signals generated by the retina so as to increase the information content at each pixel. Attribute images can be computed in parallel and exist simultaneously in registration with the original image. Pixels with similar attributes can be grouped into entities. When a pixel is grouped into an entity, it is assigned the name of the entity. This produces an entity image.

Df. An **entity image** is *a two-dimensional array of entity names.*

An entity image is an array of names that define for each pixel the entity to which that pixel belongs. Each pixel in an entity image is either contained within the entity, or contains the entity (if the image of the entity is smaller than a single pixel). An entity image is much like a

"paint by numbers" drawing where each region in the image contains the name (or identification number) of the color to be painted into that region.

The name assigned to each pixel in an entity image is a pointer to an entity frame that contains the attributes of the entity. Entity attributes are characteristics and properties of the whole region occupied by the entity. Entity attributes can be computed by integrating the attributes of the pixels belonging to the entity [Adelson and Movshon82].

Entities with similar attributes can be grouped into higher-level entities. Grouping is inherently a hierarchical process that occurs at every level of the sensory processing hierarchy. Pixels can be grouped into list entities. List entities can be grouped into surface entities. Surface entities can be grouped into object entities. Object entities can be grouped into group entities. At each level of grouping, each pixel acquires a new entity pointer that points to an entity frame. Thus, each pixel acquires a set of entity names that point to the set of entity frames to which the pixel belongs. This produces a set of entity images as shown in Figure 21.

An entity image segments the input image into a set of regions, or windows, over which entity attributes can be computed. Computed entity attributes can then be compared with class attributes stored in a library of class prototypes in the KD. When computed entity attributes match the attributes of a class prototype, the observed entity can be classified or recognized as a member of the class. When an entity is classified, the pixels that are labeled with the entity name can be assigned the class name of the entity class. This produces a set of class images as shown in Figure 21.

Df. A **class image** is *a two dimensional array of entity class names.*

An entity class name specifies, for each region in an entity image, the class of the entity that occupies that region. Class images may designate generic or specific classes, or both.

Df. A **value image** is a *two dimensional array of cost/risk/worth values.*

Entity classes can be assigned values that specify the worth of entities in that class, or the cost or risk of carrying out action on them. Cost/risk/worth values of entities can be assigned to the pixels contained in the entity class image. This generates a set of value images such as shown in Figure 21. These can be used for planning behaviors in the image or map domain. For example, worth values assigned to entity classes can be used to define priorities for tasks designed to acquire them. Cost/risk values assigned to regions of terrain can be used for planning paths that traverse them. Worth assigned to assets can define how much they are worth to defend. Worth assigned to targets can specify how valuable they would be to destroy. Worth assigned to opponents can define how much it is worth to defeat them. Value images enable plans to be formulated in the image domain using the mathematics of computational geometry [Koenderink90]

4.4.5 Images and Frames

Figure 22 shows the relationship between entity images and entity frames. On the left is a set of attribute and entity images. A surface entity image is shown that contains three surfaces (including the background). For each pixel in the surface entity image, there is a pointer to the surface entity frame to which the pixel belongs. There also exists a pointer from each entity frame back to the entity image in which the pixels that belong to that entity are located. This two-way linkage between entity images and entity frames enables computation in both the symbolic and image domains. For example, an entity image can be used as a mask or window for focusing attention or correlating observed entity attributes with predicted entity attributes. An entity image can also be used as an integration window for integrating pixel attributes to obtain entity attributes.

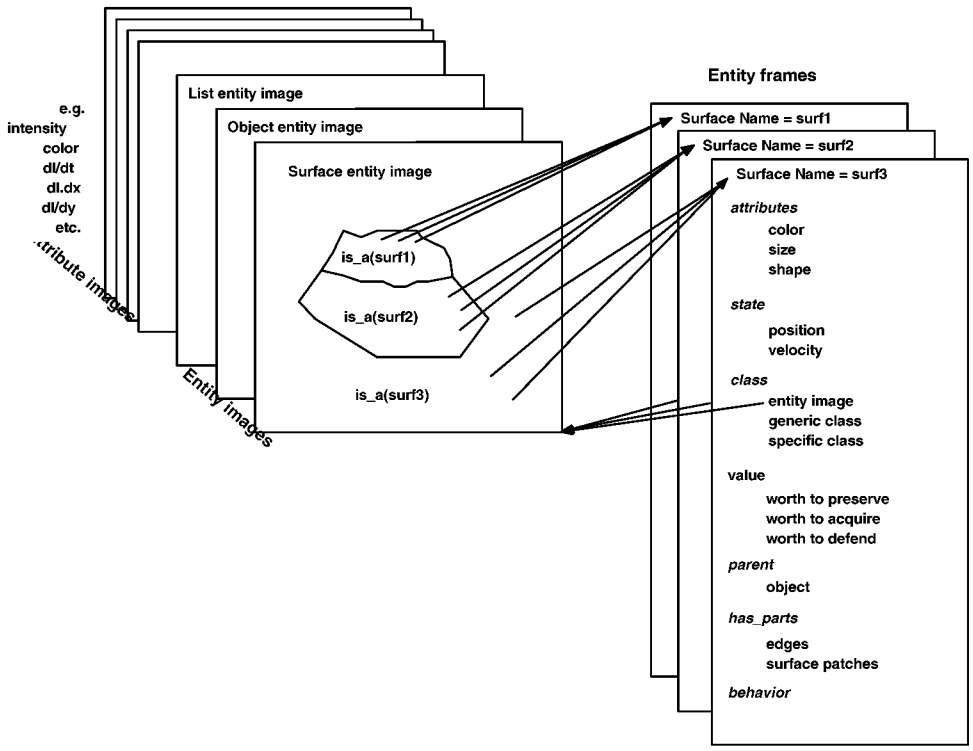


Figure 22. Relationship between entity images and entity frames. In this example, a pointer for each pixel in a surface entity image points to the frame of the surface entity to which that pixel belongs. The entity frame contains properties of the entity including attributes, state, class, value, relationships, and behavior.

If a pixel is not assigned to an entity, it can be designated as a background pixel, or as unknown. Background pixels are those that require little or no additional processing. Unknown pixels may or may not be selected for additional processing, depending on whether the contents of the region of space in the world that they represent is considered to be important to the current task.

4.4.6 Image Field of View

A TV camera has a field of view that is determined by the focal length of the lens and the size of the photosensitive array. The image resolution is defined by the number of photosenors in the array and by the quality of the optics. A two dimensional image array can be represented as a flat plane such as a typical photograph or the view through a flat window pane. An image can also be represented as a section of a spherical surface seen from the center of the sphere such as a portrayal of the sky in a planetarium, or the actual view of the sky on a clear night.

For a small field of view, there is little difference between a planar or spherical representation. However, for wide field of view images, the spherical egosphere representation is preferable to the flat plane representation. For a vision system on an unmanned ground vehicle that may need to pay attention to entities and events in many different directions (front, back, right, left, and overhead), a planar coordinate system has several undesirable properties. A planar representation goes to infinity at ninety degrees to the right and left (or up and down), and cannot represent the hemisphere behind the camera at all. On the other hand, the egosphere is continuous and isotropic in all directions. For example, it is easy and natural to represent the stars and planets in the sky at night as if they were etched on a celestial sphere. Astronomers and navigators routinely use the celestial sphere representation for mapping the heavens and computing position on the earth from star sightings in the sky.

Df. An **egosphere** is a *spherical coordinate system with the self (ego) at the origin.*

The egosphere is the most intuitive of all coordinate systems. Each of us resides at the origin of our own egosphere. Everything that we observe in the world can be described as being located at some azimuth, elevation, and range measured from the center of ourselves. To the observer at the center of the egosphere, the world is seen as if through a transparent sphere. Each observed point in the world appears on the egosphere at a location defined by the azimuth and elevation of that point. There is, in fact, no way for a single stationary eye to tell whether a scene being viewed is real, or an image projected on an egosphere with resolution equal to or better than that of the eye.

The orientation of the egosphere can be uniquely defined by specifying two orthogonal directions. Typically these are chosen to define the pole (which in turn defines an equator 90° away), and a zero azimuth on the equator. A number of different egosphere coordinate frames are useful for representing the world. These include the sensor egosphere, the head egosphere, the body egosphere, the inertial egosphere, and the velocity egosphere. [Gibson50]

Df. A **sensor egosphere** is an *egosphere where the horizontal axis of the sensor array defines the equator of the egosphere and hence the pole. The center pixel in the sensor array defines a zero azimuth on the equator.*

Figure 23 shows a sensor egosphere for a TV camera. The optical axis defines zero azimuth and elevation on the egosphere. The gray area around the optical axis is the field of view. The equator of the egosphere is defined by the horizontal line of pixels through the center of the image. The x-axis is horizontal to the right in the image. The North pole of the egosphere

is defined by "up" in the image. The position of each pixel in the image is defined by its azimuth and elevation on the egosphere. Each pixel can be assigned a range attribute whose value is the estimated distance to the point from the center of the egosphere.

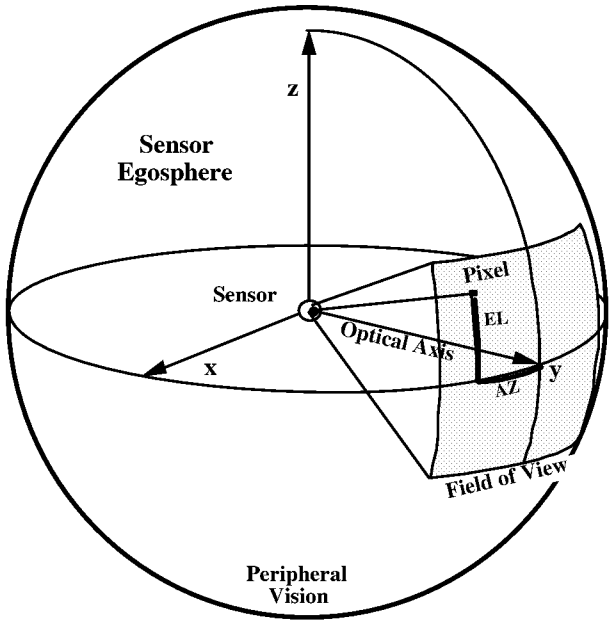


Figure 23. A sensor egosphere for a camera. The center of the imaging lens lies at the center of the sphere. The optical axis of the camera defines the zero point on the equator. The orientation of the photodetector array defines the z-axis. Each pixel has a unique azimuth and elevation. The field of view of the array covers a region on the egosphere.

Each eye has its own egosphere. Attributes measured by each eye can be transformed into a head egosphere by one translation and one rotation. Head egosphere coordinates can be transformed into torso and limb coordinates, and then into muscle coordinates by additional translations and rotations. All of these transformations are well defined so long as there is a good range estimate at each pixel. Dickmanns [99] describes the various coordinate frames and transformations relevant to a vision system used for driving an autonomous vehicle.

Range can be measured directly by LADAR, radar, or sonar; or can be computed from a wide variety of parameters including ocular vergence, stereo disparity, motion parallax, image flow, shading, texture, occlusion boundaries, position in the image, and estimates of size and speed.

For an acoustic system, the direction, intensity, frequency, and other attributes of incoming sound can be represented on the head egosphere. For tactile sensors, the location and qualities of sensed surfaces can be plotted in head egosphere coordinates. The head egosphere thus provides a convenient coordinate frame for fusion of information from vision, hearing, and touch.

A polar egosphere coordinate system suffers from singular points at the poles. The effect of these singularities can be minimized by placing the poles on the vertical axis so that they are

far from the center of the field of view. On the other hand, the pole singularities can sometimes be used to advantage, as in the case of the *velocity egosphere*.

Df. A **velocity egosphere** is an *egosphere where the velocity vector defines the pole of the egosphere and azimuth is defined relative to the current acceleration vector.*

Figure 24 shows a velocity egosphere. The velocity vector defines the pole of the velocity egosphere. The projected image of each point lying on a stationary surface in the world moves along a great circle arc radiating from the pole of the velocity egosphere.

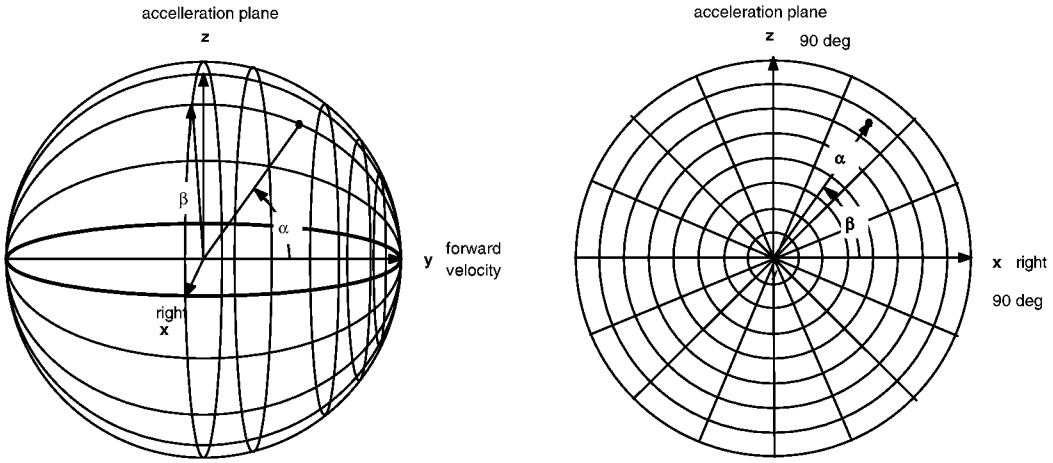


Figure 24. Two views of a velocity egosphere. On the left is a perspective view from the upper right side. On the right is a view along the velocity vector from the center. The velocity vector defines the pole of the velocity egosphere, and that corresponds to the focus of expansion of flow vectors for stationary points in the image.

For constant velocity, the images of stationary points in the environment move on the velocity egosphere at an angular rate given by the simple formula [Gibson, Olum, and Rosenblatt 1955]:

$$d\alpha/dt = (v \sin \alpha) / r$$
$$d\beta/dt = 0$$

where:
α = the angle between the velocity vector and the pixel on the velocity egosphere
v = velocity of the camera through the world
r = range to the point in the world
β = azimuth of the pixel relative to the intersection between the plane normal to the acceleration vector and the equator of the velocity egosphere

Note the simple relationship between range, velocity, and α. Note also that this relationship is independent of β. This simplifies the problem of distinguishing between optical flow resulting from self motion and optical flow resulting from object motion.

Additional egosphere representations are the *body and inertial egospheres.*

Df. A **body egosphere** is an *egosphere where the roll axis of the body defines zero azimuth and elevation, and the yaw axis defines the pole.*

Df. An **inertial egosphere** is an *egosphere where the Earth's gravity vector defines the equator and North defines a zero azimuth.*

The body egosphere is useful for local path planning for mobility or manipulation. The inertial egosphere is useful for global path planning and for transformations to and from world map coordinates. The inertial egosphere provides a stabilized internal representation of the world independent of rotation of the sensory array. Figure 25 is a top view of several egosphere representations showing the difference in azimuth between the head egosphere, the head-inertial egosphere, the sensor egosphere, a pixel ray, and the self velocity vector.

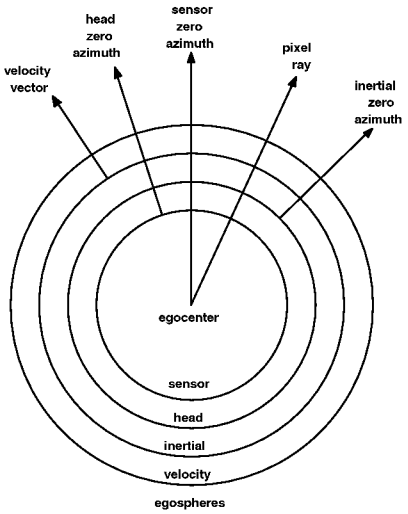


Figure 25. A 2-D top down projection of four egosphere representations illustrating angular relationships between egospheres. Pixels are represented on each egosphere such that images remain in registration. Pixel attributes detected on one egosphere may thus be inherited on others. It should be noted that pixel resolution is not typically uniform on a single egosphere, nor is it necessarily the same for different egospheres, or even for different attributes on the same egosphere [Albus 90].

Figure 26 shows a Mercator projection of the human head egosphere with the field of view of both right and left eyes shown. The foveas of the two eyes are converged on the top of a rock directly ahead. Central vision for both eyes overlaps and provides good stereo depth perception for objects less than about 50 meters (m) away. This means that range measurements are available for those pixels within central vision. Peripheral vision does not completely overlap, due mostly to blockage of the contra-lateral field of view by the nose. Near the edges of peripheral vision, resolution is low and recognition based on color and shape is poor, but perception of motion remains good. The example in Figure 26 is for a vehicle driving task. The rock directly ahead is a potential obstacle that needs to be examined. Thus, the high-resolution foveal regions of both eyes are converged on the highest point of the rock. This enables high magnification and good range measurements of the top of the rock.

In Figure 26, the size of the human fovea relative to the peripheral field of view and to the entire egosphere is to scale. This illustrates that the fovea is essentially an optical probe that is pointed by an attention mechanism at the current most important part of the world. About one third of the pixels in the entire visual field are concentrated in the fovea. A second third of the pixels fill the region marked as central vision. The remaining third are distributed over the peripheral visual field. The relative density of pixels in human vision decreases roughly exponentially with the angle between the fovea and a pixel.

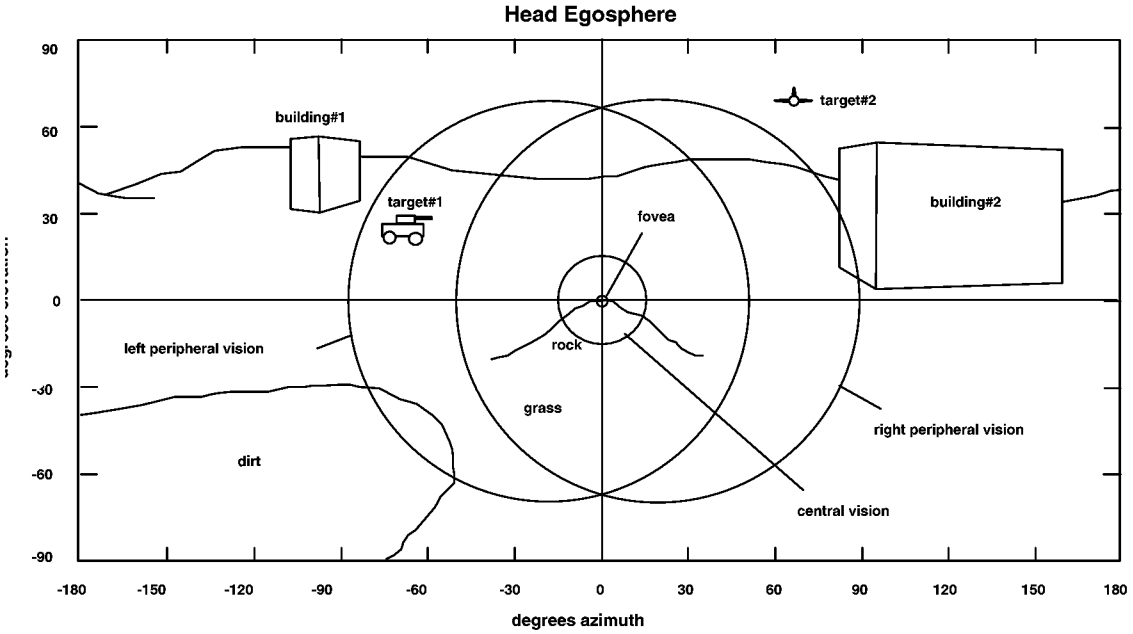


Figure 26. A Mercator projection of the head egosphere showing the field of view of foveal and peripheral vision and a number of objects in the world. The two eyes are verged on the top of the rock straight ahead.

4.4.7 Maps

A map is a two-dimensional array of attributes and entities that are scaled to, and registered with, known locations in the world.

A world map consists of a vertical projection of the surface of the earth onto the planar surface of a world map. When the position and orientation of the self is known in the world and range can be measured or estimated at each point in the visual field, it is possible to transform images from egosphere coordinates to egocentric map coordinates, and thence to world map coordinates. This makes it possible for entities observed in the image to be overlaid on a priori information in the world map. For example, images acquired by cameras are often transformed into world map coordinates for planning routes.

If terrain elevation data is available in the world map, it is also possible to transform world map information back into egosphere coordinates. Typical world map coordinates are latitude and longitude. World map coordinates have the advantage that stationary features on the ground do not move on the map when the self moves. Instead, the location of the self moves on the map. Thus, when stationary objects observed on the egosphere are transformed into world map coordinates, they become stationary on the map regardless of motion of the self. Thus, even though images of stationary objects move over the egosphere as the self moves through the world, repeated sightings of stationary objects from various view points can be integrated into a single representation on the world map. For this reason, images from cameras are typically transformed into world map coordinates to facilitate global planning of routes for moving through the world.

Attributes of map pixels may describe the characteristics of the terrain contained within the pixel. Map pixel attributes may include type of ground cover, terrain elevation relative to the sea level or relative to the vehicle, and terrain roughness, slope, and traversability. Map pixels may also include names of, or pointers to, icons that represent entities such as buildings, roads, bodies of water, or landmarks that the pixel covers, or of which the pixel is a part. Maps typically have a number of overlays that are registered with the map. Each overlay may represent one or more attribute or entity maps [Tomlin90]. For example, map overlays may represent roads and towns, rivers and lakes, buildings and landmarks. They may also represent topographic elevation and contours of constant elevation, as well as fields, woods, swamps, water, and sand. Map overlays can represent the surface roughness, the slope, or traversability as a function of direction at each pixel, and which regions are visible from a particular viewpoint. Battlefield map overlays can represent the deployment of friendly and enemy troops, the coverage of artillery, the location of mines, or the position of sector lines.

Military maps with multiple overlays are typically made available to soldiers as prior knowledge for military missions. However, a priori maps are too low in resolution to be used by autonomous vehicles for obstacle avoidance, and too static and stale to be used to analyze and anticipate the behavior of moving targets. Most military maps have elevation postings at 30 m grid intervals. In special cases, maps with elevation postings every 3 m or even every one meter may be available, and vector representation of features such as buildings, roads, and landmarks may be included with greater precision. But the time delay between when the terrain is sensed and the maps are available in the world model for decision making and control is at best minutes to hours, and more often days to weeks. A priori maps cannot provide the dynamic information necessary to track and predict moving objects in the world.

High resolution dynamic information must be generated from real-time sensory data. In the vicinity of the cameras, LADAR and stereo systems can provide range information in egosphere coordinates accurate to a few centimeters. This information can be used to build local terrain maps in real time and to represent moving objects. When the position and orientation of the camera egosphere is known, local maps generated from camera data can be registered with a priori maps. This enables landmark recognition and provides the information needed for path planning and task decomposition. Images in egosphere coordinates can be transformed into maps in world coordinates for route selection, path planning, and obstacle avoidance. A priori maps can also be transformed into egosphere coordinates so that terrain features and elevation

can be transformed into a sensor viewpoint so that names and attributes of entities are overlaid on camera images. For example, the name of a landmark or a road can be overlaid on the image of the landmark or road. Attributes of a region or entity, such as its traversability, or its value as a target or a resource to defend, can be overlaid on the image of the region or entity. This enables image based task planning and target selection to be performed directly in egosphere coordinates.

There are many issues related to the computational power required for real-time image processing, and for transformation of pixels from image coordinates to map coordinates and vice versa. The 4D/RCS reference model architecture addresses these issues by limiting the range and resolution of images and maps in the world model at each level of the hierarchy, and by focusing attention on important regions on the egosphere. Both of these approaches reduce the number of map pixels that require updating at each level. At each hierarchical level, maps have less resolution but greater range than at the level below. At each level, maps have more resolution but less range than at the level above. RCS methodology uses a heuristic (rule of thumb) that range increases about an order of magnitude and resolution decreases by an order of magnitude at each higher level. Thus, the information density on maps in the world model remains relatively constant across levels. This produces an exponential increase in resolution and decrease in range in the space-time egosphere that can be seen in the planning maps shown in Figure 7. Planned waypoints at higher levels are larger (lower resolution) and further apart (longer range) than planned waypoints at lower levels. Thus, plans for actions that are closer in space-time are more detailed and immediate than those that are more distant in space-time.

4D/RCS also supports the use of foveal/peripheral cameras to limit the number of pixels in the image. For example, it is possible to represent a full 4 π steradian egosphere field of view with resolution of 1° per pixel with less than 42,000 pixels. A 32 x 32° field of view can be represented by a 256x256 image with resolution of 0.12° per pixel. A 4 x 4° field of view can be represented by a 256x256 image with resolution of 0.016° per pixel. This is better than human 20/20 vision. A LADAR might have only about 8000 pixels with resolution of about 0.5° per pixel. A radar map of the ground may contain about 30,000 pixels. All of these egosphere representations put together require less than the number of pixels in a single 512x512 image.

Moreover, not all pixels need to be processed in real time. Attention mechanisms can mask out pixels that are irrelevant to behavioral goals. This may reduce the number of pixels that require real time processing by as much as an order of magnitude. Thus, the total number of image pixels that must be processed at frame rates may be less than 100,000. To achieve real-time processing rates of ten frames per second may require processing of less than a million pixels per second. For modern image processing technologies, this is not an impractically large computational load.

4.4.8 Events

Df. An **event** is *something with distinctive temporal structure that occurs during a period of time.*

In computer science, switching theory, and discrete event systems, an event is defined as a change in state that occurs instantaneously in zero time. However, an instantaneous event is an artificial construct designed to simplify mathematical computation. In the real world nothing occurs instantaneously, albeit time intervals can be as short as nature mandates. State changes in atoms and molecules may occur in picoseconds. State changes in electronic circuits occur in nanoseconds. State changes in mechanical systems occur over periods of milliseconds or seconds. An event such as an earthquake may occur over a period of minutes. An event such as a wedding or a funeral may occur over a period of an hour. An event such as a concert or ball game may take several hours. An event such as leaves turning color in the fall may take place over days or weeks. An event such as the decline of the Roman Empire may occur over a period of more than a century. An event such as the extinction of the dinosaurs may take thousands of years. The common feature among all events is that they occur over an interval in time and involve some change in state.

When we observe the world we perceive events. With our ears we perceive clicks, notes, words, and melodies. We perceive birds singing, cats meowing, dogs barking, crickets chirping, babies crying. We perceive traffic noises. We attend concerts and listen to lectures and debates. With our eyes we perceive patterns of movement. We see waves on the water. We see trees swaying in the breeze. We see patterns of motion in the ballet, in sporting events, and in human gestures. These patterns of movement can be temporally grouped and segmented into events. Events can have names and attributes and can be related to other events. Patterns of events can be grouped into higher level events. Just as entities can be represented symbolically by entity frames, so events can be represented symbolically by event frames.

What an entity is to space, an event is to time. An entity occupies a region in space. An event occupies an interval in time. An event may occur at a point or during a period of time, depending upon the resolution of the clock. For example, the event of a light bulb turning on takes place over a period of tens of milliseconds as the filament heats up. The light bulb can be considered to turn on at a point in time if the system has a clock that samples the environment only once per second. However, if the system samples every millisecond, the light bulb turning on is an event that takes place over an interval, a number of sample periods.

Just as there are external and internal entities, there are external and internal events. An external event is a situation or state change that occurs in the real world. Examples include a rifle shot, a tree falling, an airplane crash, and a performance of a maneuver or tactic. An internal event is a data structure in the observer's knowledge database. An internal event may be the detection of a muzzle flash, the recognition of an acoustic signature, a data structure representing a relationship between objects in an image, or a state variable indicating the achievement of a goal in a task. Just as entities can be represented symbolically by entity frames, so events can be represented symbolically by event frames.

Df. An **event frame** is *a data structure that contains the event name, a list of attribute-value pairs, a pointer to a parent event, and pointers to subevents.*

An event frame contains the information that the intelligent system knows about an event. Figure 27 shows an example of an event frame.

```
NAME  = event_id (uncertainty)          // this is the frame address in the KD

// attributes  -  these are characteristics that address the question What?
amplitude     = magnitude of variations
shape         = time-frequency pattern, amplitude waveform, symmetry
spectrum      = Fourier transform over the event

// state – these are dynamic properties that address the question When?
boundaries    = start-end times
trigger       = proximal cause of the event
length        = duration of the event

// class – pointer to the channel and class or classes to which the event belongs
channel           = signal pathway where the event was detected
generic_class1    = pointer to generic1 class exemplar
generic_class2    = pointer to generic2 class exemplar
specific_class    = pointer to specific class exemplar

// value – worth of the event
benefit of event  = value
cost of event     = value

// pointers that define grouping relationships
belongs_to        = parent event
has_part          = subevent1
has_part          = subevent2
has_part          = subevent3

// pointers that define situational relationships
prior state   = conditions prior to the event
effect        = conditions resulting from the event
participants  = entities involved in or affected by the event

// functions that define behavior
responds_to
acts_like
```

Figure 27. The structure of a typical event frame. Each event frame consists of a name, a list of attributes, and a set of pointers to other events.

The event name is an address or index by which the event frame can be accessed in a database or library of events, or to which other event and entity frames can be linked. An uncertainty parameter can be associated with the name to indicate how certain the system is that the event has been properly identified. The event frame contains event attributes that describe attributes such as amplitude, shape, and spectrum. Event attributes may also characterize the event as an impulse, a string of impulses, a frequency, or a pattern of frequencies on a channel or set of channels.

Event state-variables describe dynamic properties such as start-end times, trigger, and length of the event. Uncertainty parameters can be associated with state parameters to indicate how dependable the estimates of these values are.

The event frame may contain pointers to the signal channel on which the event was detected and to the class or classes to which the event belongs. For example, an event may have a generic class pointer that identifies it as a click, note, word, or melody. Generic events are events that have been classified but are not unique. Examples of generic event classes include

shots, chirps, words, speeches, ball games, and wars. Generic classes of events include point samples of sensory signals, and strings of samples that make tones. A fork event might be a point in time when two instruments begin to play two voices. A joint event is a point where a duet becomes a solo. A time-frequency patch might be a sonogram of a musical chord or a speech phoneme. A boundary might be a partition between words or notes. A time-frequency surface might be a portion of an orchestral performance in which many instruments are playing many different parts simultaneously over a period of time. A specific class pointer may identify it as a particular word or melody. Examples of specific events include the shot that killed JFK, the string of words spoken by Neil Armstrong when he first set foot on the moon, and a specific concert by the Boston Pops.

The event frame may contain value attributes that define the cost or benefit cost of the event. The event frame may contain pointers that define inheritance relationships with other events. Each event frame has a pointer to the frame of a parent event of which it is a part, and a set of pointers to frames of sub-events that are its parts. For example, a word event may have a parent_pointer to the phrase or sentence to which the word belongs. It will have a number of subevent_pointers to the frames of the phoneme that are its parts. Inheritance pointers are established and maintained by grouping and classification operations performed by sensory processing functions.

The event frame may also contain pointers that define situational relationships such as prior conditions, effect, and entities that are involved in or affected by the event. Event effects may include changes in state produced by the event. A state change may be a change in position, orientation, velocity, color, temperature, size, or shape of entities in the world. State changes may also be changes in a signal from a sensor, or the occurrence of a string or pattern of signals. It should be noted that two or more events may result in no net change in state. For example, the signal on a wire may change state from a "0" to a "1", and then change back to a "0". The net change is zero, but the historical record contains two transition events (or one "square pulse" event.)

An event frame may include a set of functions that describe what causes the event to occur under certain conditions or in response to certain stimuli. Behavioral functions and parameters may be inherited from generic or specific class prototypes.

Event frames contain pointers that define a taxonomy of grouping relationships. Each event frame has a pointer to the frame of a parent event of which it is a part, or to which it belongs. Each event frame also contains a set of pointers to frames of subevents that are its parts, or that belong to it. A level 4 event frame has a parent_pointer to the level 5 event frame to which it belongs. It has a number of subevent_pointers to the level 3 event frames that are its parts. Each level 3 event frame has a parent_pointer to the level 4 event frame to which it belongs, and a set of subevent_pointers to the level 2 event frames that are its parts. Each level 2 event frame has a parent_pointer to the level 3 event frame to which it belongs, and a set of subevent_pointers to the level 1 events that are its parts. Each level 1 event has a parent_pointer to the level 2 event frame to which it belongs. All of these pointers are established and maintained by grouping operations that are performed by sensory processing functions discussed later.

Event frames may stand alone as data structures, or be linked to form lists, strings, words, sentences, networks, and maps. Named event frames may be used to describe what happens in the world. In language applications, event attributes may serve as adjectives that describe characteristics of the events to which they are attached.

Event frames can be interconnected by pointers to form causal, semantic, and situational networks. Situational networks may represent situations or temporal relationships such as "before," "after," or "simultaneous-with." Situational networks may also have pointers to maps or images that pictorially display temporal relationships. Causal networks represent the cause-and-effect relationships between events, situations, and actions that occur in the world. Semantic networks represent the relationships between events, attributes, situations, and actions. Semantic networks define meaning and enable reasoning, logic, and language.

4.4.9 Event Grouping

Just as subentities can be grouped into entities in space, subevents can be grouped into events in time. For example, a series of acoustic signal values can be grouped into a phoneme. A series of phonemes can be grouped into a word. A series of words can be grouped into a sentence. A series of sentences can be grouped into a paragraph, or concept in a speech. Figure 28 illustrates the temporal grouping of subevents into events.

Event frame grouping defines temporal relationships such as "before," "after," and "simultaneous." Events frequently serve as triggers for action on the part of an intelligent system. The interconnecting pointers between event frames and entity frames can produce semantic, causal, or situational networks that define a historical record. Semantic and causal networks define meaning and causality.

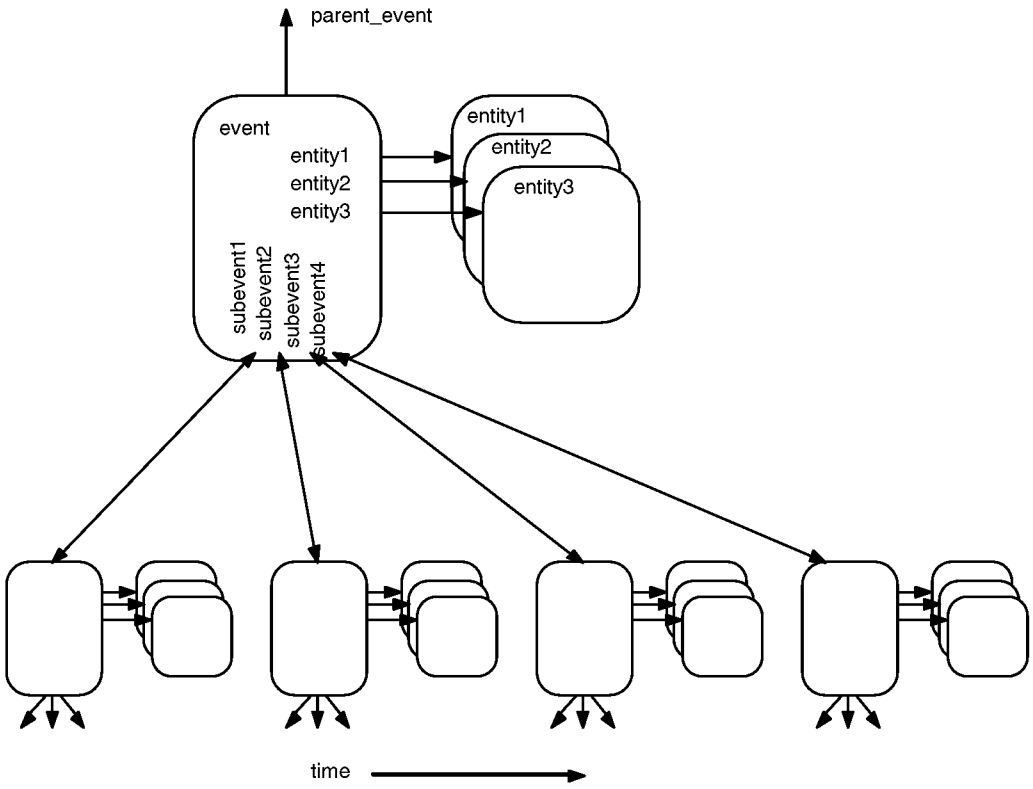


Figure 28. Grouping of subevents into events along the time line. Event frames contain pointers that define a hierarchy of grouping relationships. The event grouping hypothesis generates for each event frame a set of pointers to frames of subevents that are its parts, or that belong to it. Event grouping also produces for each event frame a pointer to the parent event of which it is a part, or to which it belongs. Each event frame may also contain pointers to entities that participate in the event.

4.4.10 Timing

Figure 29 is a diagram of the temporal relationships involved in representing the time line. The most obvious thing about time is that is flows in one direction and there is a unique point called the present that divides the past from the future. We can choose the origin of the time line (where t = 0) to be the present. To the right is the future. To the left is the past. The present is the 0^{th} tick of the clock that separates the future from the past.

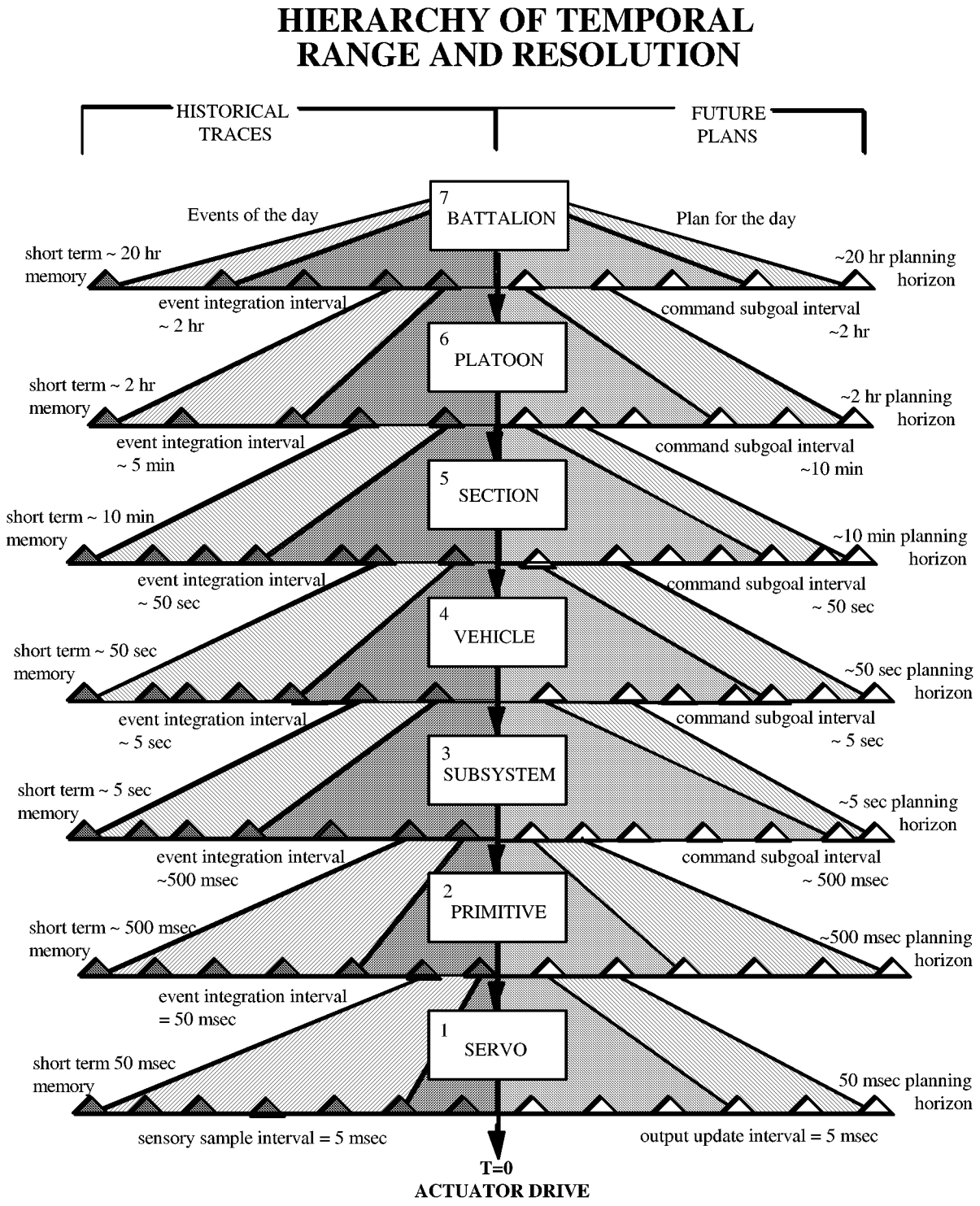


Figure 29. A timing diagram for 4D/RCS showing the temporal range and resolution for planning and short term memory at different hierarchical levels. Planned events are indicated by the intervals between open triangles on the right. Detected events are indicated by intervals between filled triangles on the left.

It can be observed in Figure 29 that there is approximate symmetry between future plans and historical traces. At each level, tasks are decomposed into plans consisting of a string of subtasks that take place over time. Each task begins with a starting state and terminates when the goal is achieved. Similarly, at each level, a string of subevents that take place over time are grouped into events. Each event begins at a point in time and terminates when the event is complete. As time flows, task plans are executed and events are observed.

In the 4D/RCS architecture, different levels in the hierarchy represent knowledge with different range and resolution in both space and time. At each level, subevents are grouped into events along the time line in much the same way that subentities are grouped into entities in the spatial domain. As a rule of thumb based on symmetry, the historical period over which events are stored in short term memory is roughly equal to the planning horizon over which plans are made for the future. At each level, the resolution in time is such that less than ten subgoals for each agent are required from the real-time planner. Similarly, the resolution of the historical trace is such that only about ten subevents from each observer are grouped into each event.

At the input to the servo level, each sample of a signal is an event. At the output of the servo level, an event might be the detection of a phoneme, or motion in an image. At the output of the primitive level, an event might be the detection of an acoustic signature, or the recognition of a pattern of movement in an image. At the output of the subsystem level, an event might be the traversal of an intersection or the avoidance of an obstacle. At the output of the vehicle level, an event might be the crossing of a bridge or the entering of a village. At the output of the section level, an event may be the crossing of a valley, or the establishment of an observation post.

4.4.11 Temporal Persistence of Representation

In order for an intelligent system to make the most efficient use of the available time, computational power, and memory capacity, it is useful to partition the knowledge database into at least three parts. The first is immediate sensory experience. It consists of the current input, the current estimated state and attributes of entities, and predictions based on the current estimated state. The second is short-term memory. It consists of a historical record of the recent past. At each level of the 4D/RCS hierarchy, a record of events is preserved that extends into the past about as far as the planning horizon extends into the future. The third is long-term memory. It consists of the entire store of knowledge acquired by the intelligent system over its lifetime.

Immediate Experience

Immediate experience is rich, vivid, and dynamic. Visual images of the real world are complex and full of color and motion with three-dimensional perspective. Auditory experiences have a wide range of frequency and intensity attributes with a sense of direction based on amplitude and phase differences between the two ears. Tactile experiences can convey rich sensations of temperature, vibration, pressure, and texture that, when combined with proprioception, can provide a sense of position, velocity, and force.

However, the richness of detail in the immediate experience is transient. It disappears as soon as the camera switches viewpoints, or the sound stops, or tactile sensors lose contact with the environment. All that remains of immediate sensory experience after the stimulus is removed are memories consisting of symbolic information that was specifically noticed during the experience and transferred into short-term memory.

The data structures that support immediate experience are those necessary for image processing, signal detection, recursive estimation, and model-based predictive filtering. These include:

- **Observed signals, attribute images, and states.** These consist of the current array of signals and images from sensors and the attributes and states that can be directly computed from sensory signals.
- **Estimated signals, attribute images, and states.** These are the output of filtering processes that integrate information from observed images and other sources over the filter space-time interval.
- **Predicted signals, images, attributes, and states.** These are the system's best guess of what the next sensory input of signals and images will be, based on the latest estimated state-variables, the known system dynamics, and the effects of known control outputs.

These representations of immediate experience exist concurrently with the sensory input and disappear with it as well. None of these representations persist for more than a few milliseconds after the stimulus ceases.

Short-term Memory

Short-term memory differs from immediate experience in that it provides a temporal recirculating record of events that persists for some interval after the sensory input disappears. Short-term memory can store sequences such as a series of events, a string of words, an acoustic signature, the path of an entity through space, or the trajectory of an attribute vector through state-space. Short-term memory provides the temporary storage required for processing strings of events, performing time and frequency analyses, and recognizing or detecting temporal patterns. At each level in the 4D/RCS hierarchy, short-term memory preserves a historical record of events that stretches into the past about as far as the planning horizon at that same level reaches into the future. This is illustrated in Figure 29. Short-term memory maintains a temporary list of events and participating entities. These constitute the current focus of attention. Entities and events can be specified by the BG process as important to the current or next planned task. Entities and events can also be flagged by SP/WM/VJ as particularly note-worthy. Entities-of-attention may be used by SP to generate spatial masks and grouping hypotheses for images. Events-of-attention can generate temporal windows and segmentation hypotheses for acoustic signals. State variables and entity images of entities-of-attention provide targets for pointing cameras and other sensors

Short-term memory differs from long-term memory in that it is dynamic. Short-term memory retains information only so long as it is relevant to the current focus of attention. When

the focus of attention shifts, short-term memory is overwritten with new information. What was previously stored in short-term memory is lost.

Entities and events in long-term memory can be selected by task-driven attention functions to be transferred into short-term memory where they become entities and events of attention. Attributes of entities and events in short-term memory can be compared with attributes of entities and events observed in immediate experience. Correlation and differences between short-term memory and immediate experience enable signal detection, recursive estimation, predictive filtering, grouping, and image segmentation processes.

Comparison between entity and event attributes in short-term memory and entity class attributes derived from long term memory also support recognition and classification processes. Classification occurs when entity attributes in geometrical entity frames match or correlate with entity attributes in entity class frames. By this means, a geometric entity or spectral event detected in immediate experience can be classified or recognized as corresponding to a generic or specific entity class or event class prototype stored in long-term memory.

Entities and events detected in immediate experience, and selected by attention functions to be transferred into short-term memory, can from there (if deemed noteworthy by the Value Judgment system) be stored in long-term memory for future use. By this process, long-term memory can be kept up-to-date. Entities and events deemed not noteworthy are lost as soon as short-term memory is overwritten by subsequent input.

Long -Term Memory

Long-term memory is a repository of information that can accumulate and be retained over indefinite periods of time. Long-term memory contains the entire dictionary of entities and events about which the intelligent system has information. Long-term memory differs from short-term memory in that information endures even through power outages or system shut down. The amount of information that can be stored in long-term memory is very large. Long-term memory is analogous to disc storage media. In biological systems, long-term memory contains only symbolic information. Images or maps are represented only roughly, if at all. Computer systems, however, are not subject to the same constraints as the biological brain. Mass storage technologies such as hard discs and CD ROMs can be used to store images in long-term computer memory. In the 4D/RCS, long-term memory will contain maps, drawings, and prototype images of generic and specific entity classes. The Demo III vehicles also sometimes carry video recording devices to store sequences of images from selected portions of missions.

Among the most important information in long-term memory for Demo III vehicles are digital terrain maps with overlays that provide information about roads, buildings, towns, rivers, and deployment of friendly and enemy forces. Using GPS or landmark recognition techniques, maps generated from sensed images can be registered and merged with stored maps for use in path planning and to assist in image processing. Local map information gathered from sensory input can also provide real-time updates to digital terrain maps stored in memory. Map updates may be shared with peer vehicles, with human operators, and with high-level digitized battlefield databases.

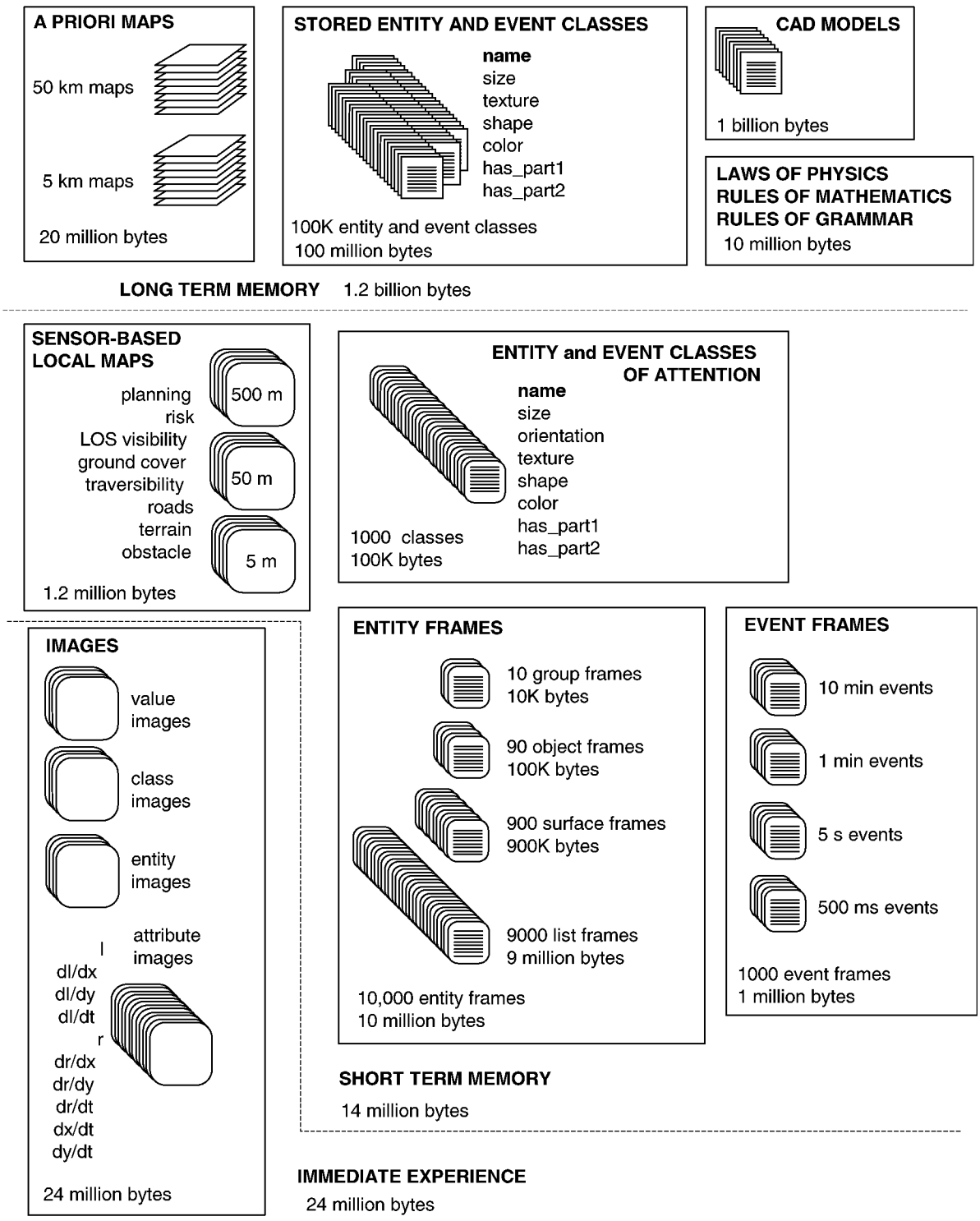


Figure 30. Data structures in immediate experience, short term, and long term memory. Images and geometric entity frames reside in immediate experience. The set of expected entity class frames reside in short term memory. The library of generic entity class frames reside in long term memory. High resolution local maps reside in short term memory and are constantly updated from immediate experience. The topographic map database resides in long term memory and is updated from short term memory only infrequently for noteworthy entities.

Long-term memory may also be used to store task knowledge such as libraries of plans, and models that can be used for simulation of plans. Figure 30 illustrates the images and geometric entity frames that are a part of immediate experience, the set of generic entity class frames that reside in short term memory, and the library of generic entity class frames in long term memory.

4.4.12 Knowledge of Rules of Mathematics and Logic

The knowledge database contains rules of mathematics and logic that can be expressed in formulae such as differential equations, production rules, statements in propositional or predicate calculus, or rules of arithmetic and geometry. These are symbolic representations that describe the way the world works and how objects, events, and actions relate to each other in time, space, causality, and probability. Most mathematical and logical functions can be expressed by rules of the form

$$\text{IF (input) THEN (output)}$$

If the input is a proposition or predicate, the output can be the truth value of the proposition or predicate. If the input is a symbolic string, the output can be another string.

If the input and output are single valued real numbers, the rule

$$\text{If (x) THEN (y)}$$

becomes

$$y = f(x)$$

If the input and output are vectors, the rule

$$\text{IF } (\mathbf{X}) \text{ THEN } (\mathbf{Y})$$

becomes

$$\mathbf{Y} = \mathbf{H}(\mathbf{X}^{\mathbf{T}})$$

where $\mathbf{H}$ is a transformation matrix.

In a digital computer, rules of mathematics and logic are typically represented in a form that can be solved by numeric or symbolic algorithms. In the brain, rules can be represented by functional mappings implemented by neural nets. In general, a function is defined as a relationship between an input and output. Thus, at least in principle, any one-to-one or many-to-one functional mapping can be represented by a table look-up where the input is the address of the location in the table where the output is stored. In other words, if the input is an address of a memory location, the output is the contents of the address. If the output is a value, then the rule returns a value. If the output is another address, then the rule returns a pointer to another location.

This suggests how rules of mathematics and logic can be implemented by neural mechanisms. The input to a neural net can be considered the address of a location in look-up table. The neural net output is then the contents of the look-up table. For example, a CMAC (Cerebellar Model Arithmetic Computer) or a multilayer neural net can implement any relatively smooth nonlinear function of a small number of input variables.[Albus75a, b]

Of course, there are practical limitations to the representation of mathematical functions by table look-up. These limitations are typically related to the dimensionality and resolution of the input space and to the precision required of the output. There are corresponding limitations to the ability of the brain to store and recall functional relationships. These limitations can sometimes be resolved by interpolation or averaging over a large population of fuzzy mappings. However, the more general solution is to invoke a numerical procedure or symbolic algorithm. Most humans solve complex problems in mathematics or logic, not by learning all possible input-output relationships, but by learning procedures to perform numerical or symbolic processes.

4.4.13 Knowledge of Rules of Physics

Rules of physics are typically expressed in mathematical formula such as differential equations that describe the relationships between force, mass, and acceleration, velocity, and position. These may be embedded in predictors, simulators, and control laws used for recursive estimation, planning, and control. Most rules of physics can also be formulated in terms of

IF (input) THEN (output)

and, hence, can be represented either by equations in digital computers or by table look-up in neural nets. For everyday experience such as running, jumping, throwing and catching a ball, or driving a car, humans learn what to expect from the environment by trial and error. This is the type of knowledge that can be stored and retrieved by neural nets in table look-up form. For more complex or less intuitive problems, procedural methods are typically invoked.

Knowledge of physics may include knowledge embedded in structural or dynamic models. Structural and dynamic models enable world modeling functions to simulate the results of hypothesized actions for planning. They also enable the world model to predict the evolution of state variables for recursive estimation and predictive filtering.

Df. Structural models are *rules and equations that describe how physical structures are kinematically connected and how forces and stresses are distributed.*

Df. Dynamic models are *rules and equations that describe how forces and inertias interact with each other in time and space.*

4.4.14 Knowledge of Value

Value is an attribute that can be assigned to the worth of entities or events. Value can also be assigned to the cost, risk, or benefit of situations, states, or plans. Value may be assigned a priori, or may be computed by cost functions that change as a function of the situation. For example, the worth value of an object may depend on its state. An object that is damaged or used may be worth less than one that is undamaged and new. The value of an event may depend on the situation in which it occurs. The sound of an approaching helicopter may be good or bad depending on whether it is a friendly or enemy aircraft. For many control problems, the value of a cost function is derived from an integral over a path through state space. The computation of cost, benefit, and risk of tentative plans enables behavior generation planners to decide how much effort to expend, or risk to accept, in attacking a target or in defending an object or position.

4.5 Sensory Processing

Within each node of the 4D/RCS architecture, sensory processing (SP) processes operate on data from sensors so that the world model (WM) processes can maintain the knowledge database (KD) as a current and accurate estimate of the state of the world, including the internal state of the system itself.

Sensory processing is organized around sensors. Each sensor produces a signal that varies in time as the physical phenomena that it measures varies in time. SP processes operate on sensory signals to window, group, filter, compare, classify, and interpret them as entities, events, and situations that correspond, in a meaningful and useful way, to entities, events, and situations in the real world. Output from SP processes is used by the WM processes to keep the KD current and up-to-date. Output from the WM may be returned to the SP processes to facilitate knowledge based windowing, grouping, filtering, comparison, classification, and interpretation of sensory input.

SP processes exist at each level of the 4D/RCS control hierarchy. The level 1 SP typically receives data from the physical sensors. The output from the lower level SPs feed into the higher level SPs for further sensory processing to produce information about the world at the corresponding levels of abstraction. In this sense, each lower level SP can be regarded as providing "virtual sensors" for the next higher level SP process.

An intelligent vehicle may have many sensors, including visual, infrared, radar, laser, acoustic, vibration, sonar, tactile, position, velocity, acceleration, force, torque, temperature, pressure, magnetic, electrical, nuclear radiation, and chemical sensors. These sensors may be grouped into a variety of sensory subsystems.

For example, vision subsystems may include black-and-white or color TV cameras, forward looking infrared (FLIR), and laser range imaging (LADAR) cameras. SP processes in visual subsystems may process images to compute brightness, color, spatial and temporal gradients, stereo disparity, range, texture, shape, and motion. This information may be used to

detect edges and boundaries, and measure the shape, position, orientation, of surfaces. The goal may be to detect obstacles, to track targets, and to recognize objects, events and situations. Camera platform encoders may measure the pointing direction of cameras relative to the vehicle, and inertial sensors may enable the camera platforms to stabilize images and determine absolute camera pointing direction and tracking velocity. This may enable SP processes to determine the position and motion of objects in the world relative to the vehicle. Characteristic size, shape, color, thermal signature, and motion enables the detection and recognition of objects, such as roads, ditches, fences, rocks, trees, bushes, dirt, sand, mud, concertino wire, fire, smoke, buildings, bodies of water, tanks, trucks, and people.

An acoustic subsystem may include microphone arrays and sonar sensors. SP processes in the acoustic subsystem may process signals to detect frequency components, compare time and frequency patterns with predicted acoustic events, measure correlations and differences, close phase-lock loops and tracking filters. This information may be used to track and recognize acoustic signatures from sources such as helicopters, jets, trucks, tanks, sniper fire, machine guns, heavy weapons, exploding ordinance, sounds of footsteps, shouts, and voices.

A mobility sensor subsystem may include accelerometers, tilt meters, gyros, global positioning satellite (GPS) receivers, odometers and speedometers to estimate position, velocity, and heading. Internal vehicle sensors may provide information about fuel levels, engine temperature, rpm, oil pressure, and vibration.

Each of these sensory subsystems has its own SP, WM, and VJ processes that extract and evaluate information from the sensory data stream that is relevant to the behavioral task being planned and executed in BG processes within the same node. As sensory data is processed, it is filtered, windowed, and combined or grouped with data from other sensors into entities, events, and situations with attributes and states that can be matched against known classes for recognition and classification. In the process, signals are transformed into symbols. Arrays of signals form images that are transformed into maps with identified regions that correspond to entities and classes in the world. Symbols and maps at a variety of hierarchical levels represent the information needed by the BG processes at each level to plan and execute behavior that maximizes the intelligent system's probability of success in accomplishing its behavioral goals.

4.5.1 Sensor Processing (SP) Functionality

Within each SP process, there are five basic processing functions.

(1) Windowing (or its inverse – masking) selects the regions of space and the segments of time over which sensory inputs will be operated on by the SP process. The remainder of the input can be masked out and ignored. The shape, position, and duration of spatial and temporal windows and masks are determined by an attention function that defines those regions in the image that are worthy of attention. The attention function selects regions that either: (a) are relevant to tasks being addressed by the BG process, or (b) contain entities or events with attributes that fall outside some expected norm or are designated as dangerous or otherwise

noteworthy. Windowed regions can be assigned priorities and be allocated computational resources in proportion to their relative importance.

(2) Grouping integrates or organizes subentities into entities and subevents into events. Grouping segments, or partitions, images into regions that can be assigned entity labels, or names, and it segments time into intervals that are assigned event labels, or names. Any particular grouping is an hypothesis based on some gestalt heuristic such as: proximity (subentities are close together in the image, or subevents are clustered along the time axis), similarity (subentities have similar attributes such as color, texture, range, or motion, or subevents have similar spectral properties or sequential relationships), continuity (subentities have directional attributes that line up, or lie on a straight line or smooth curve), and symmetry (subentities are evenly spaced or are symmetrical about a point, line, or curve).

Each entity grouping operation creates a region in an entity image consisting of a set of pixels. Each entity grouping operation also creates an entity frame that is labeled with the name of the entity. Each pixel in the entity image region has a pointer to the entity frame to which it belongs. Each entity frame contains a set of "has_part" pointers with the names of the subentities that belong to the entity. Each subentity has a "belongs_to" pointer that contains the name of the parent entity to which it belongs. Finally, each entity frame has a pointer back to the entity image that contains pixels belonging to it. These pointers are illustrated in Figure 31.

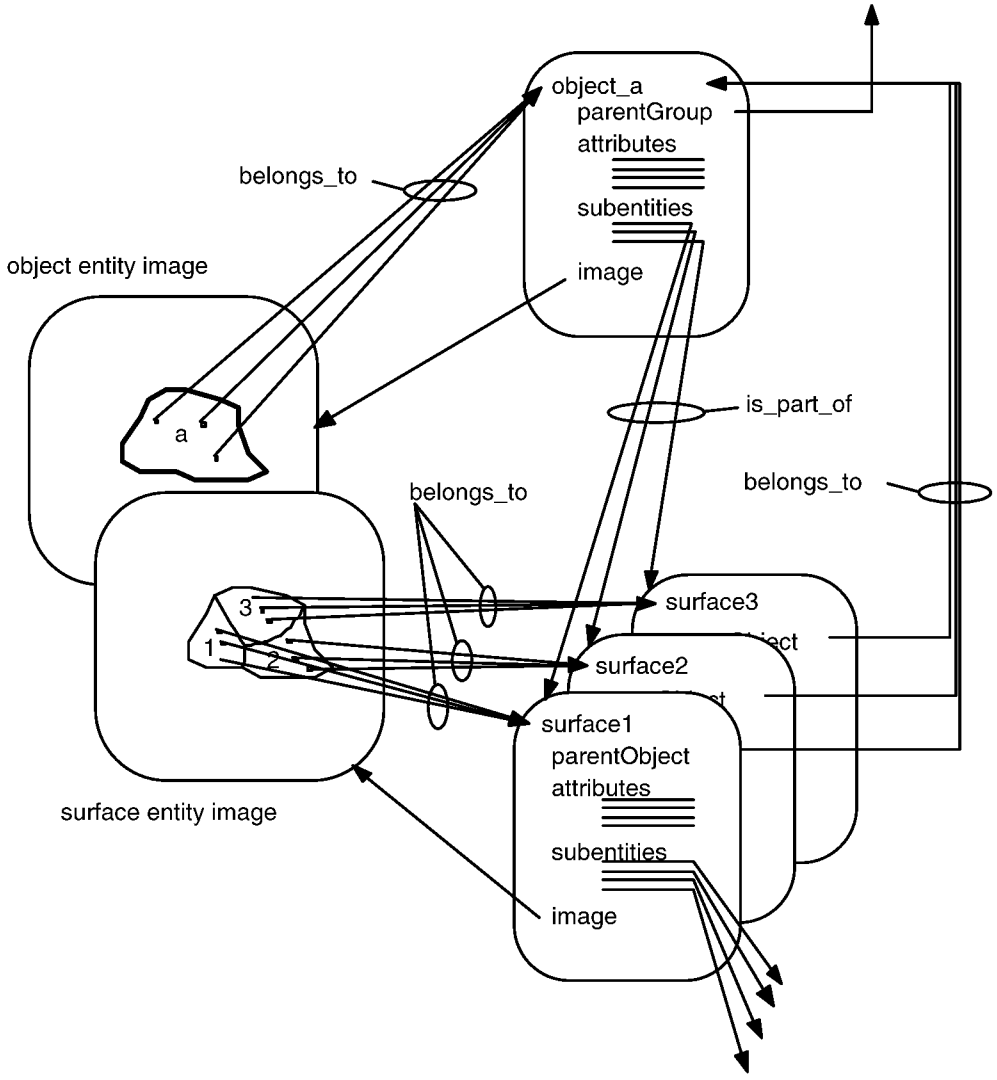


Figure 31. Pointers between entity images and entity frames generated by each grouping hypothesis. Each entity grouping hypothesis results in: (1) a segmented image in which each of the pixels in each segment has a pointer to the entity frame to which it belongs; and (2) an entity frame for each segment, with links to subentity frames, a link to a parent entity, and one or more links back to the image in which the entity appears as a segmented and labeled region.

When entity grouping is performed at multiple levels, the result is a hierarchy of entity images and entity frames such as shown in Figure 32. Event grouping at multiple levels produces a hierarchy of events such as shown in Figure 28.

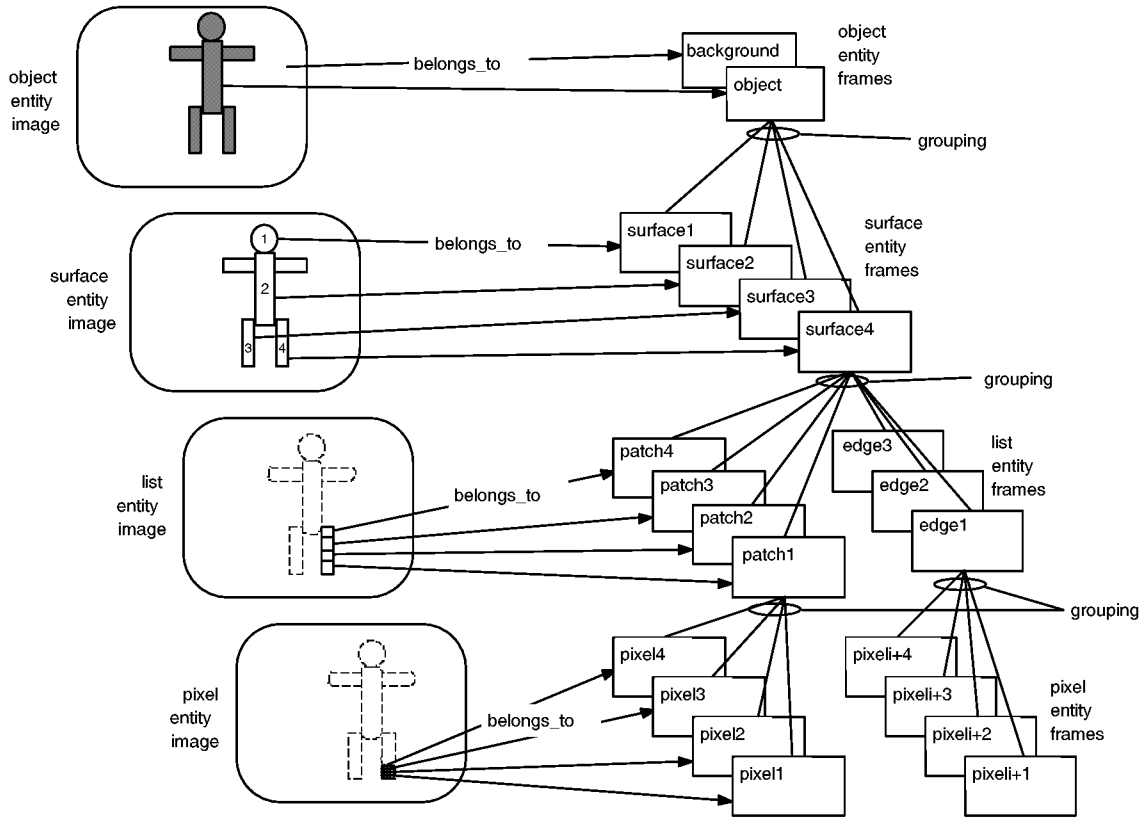


Figure 32. The network of relationships between images and entity frames established by grouping relationships at multiple levels in the 4D/RCS hierarchy.

There is, of course, no guarantee that any entity grouping hypothesis produces an internal entity with any correspondence to an entity in the real world. Two pixels that are in close proximity in an image may lie on completely different objects in the world. Two pixels of the same color and intensity may lie on different objects, and two pixels with different colors and intensities may lie on the same object. Edges that line up in an image may or may not lie on the same edge in the world. There is also no guarantee that any event grouping hypothesis produces an internal event that corresponds to an external event. Grouping hypotheses need to be tested, and confirmed or denied, by observing how well predictions based on the grouping hypothesis match subsequent observations of sensory data over time under a variety of circumstances. This is sometimes called a "reality check."

(3) Computation is a process that calculates attributes of entities and events. Event attributes such as frequency components, waveform shape, duration, and temporal pattern of motion can be computed by integrating over the duration of the event. Entity attributes such as position, velocity, orientation, area, shape, and color can be computed by integrating subentity attributes over the region of space covered by the hypothesized entity. For example, the area of a surface entity in an image can be computed by counting the number of pixels contained in it. The cross sectional area of an object entity can be computed by multiplying the area of its projection in the image by the ratio of its range to the focal length of the camera. The lateral velocity of an object entity can be computed by calculating the angular motion of the center of gravity of the object in the image and multiplying by the range to the object. The radial velocity

can be computed from range rate measurements. The orientation of an edge can be computed by regression on a line of given orientation. For surface entities, attributes such as area, texture, color, shape, and orientation can be computed. For surface boundaries entities, attributes such as length, shape, and type (color or range discontinuity) may be computed. For object entities, attributes such as size, shape, color, texture, and state including position, orientation, and velocity can be computed by integrating pixel attributes over the region occupied by the entity.

(4) Filtering is a process that reduces noise and enhances signal quality. The computed values of entity attributes can be filtered over intervals of time by recursive estimation. This accomplishes two purposes. First it computes a best estimate (over a window of space and time) of entity attribute values based on correlation and difference between predicted and observed attribute values. The resulting "best estimate" is a filtered value of the entity attribute. Second, recursive estimation generates statistical properties such as confidence factors for observed and estimated attribute values. The variance between observed and predicted attribute values can be used to compute a confidence factor for the hypothesized entity. This can be used to confirm or deny the grouping hypothesis that created the entity. If the variance between the observed and predicted attributes is small, confidence in the grouping hypothesis is increased. If the variance between the observed and predicted attributes is large, confidence is reduced. If the confidence factor for entity attributes rises above a confirmation threshold, the grouping hypothesis that generated the entity is confirmed. If the confidence falls below a denial threshold, the grouping hypothesis is denied and a new grouping hypothesis must be selected.

(5) Classification (or Recognition) is a process that establishes a match between confirmed entities and entity class prototypes stored in the system's knowledge database. Classification usually refers to assigning an observed entity to a generic entity class. Recognition usually refers to assigning an observed entity to a specific entity class. If the attributes of both an entity and a class prototype are expressed as vectors with corresponding elements, then their similarity can be computed by taking the dot product between the two attribute vectors. If the degree of similarity between the entity attribute vector and a class prototype vector exceeds a recognition threshold, the entity is assigned membership in the class.

Classification generates for each geometric entity frame a pointer that contains the name of the entity class to which that frame belongs. Classification also generates for each pixel in the entity image, a pointer that contains the name of the entity class to which it belongs. Thus, for each classified entity, the entity image is labeled with the name of the entity class. Thus image and map overlays can be constructed with pixels that are labeled as roads, buildings, bridges, targets, friendly forces, and enemy positions.

Once images and maps are labeled with entity names, additional overlays can be generated that contain the attributes of the recognized entities. Images and maps can have regions labeled with attributes such as traversability, risk, and priority or value for attack or defense. Maps with such labels are what are often needed by the BG hierarchy for planning military missions and for developing strategies, tactics, and maneuvers to accomplish those missions.

Classification of events is also based on matching event attributes with the attributes of event class prototypes. Classification establishes a match between an observed event and event classes stored in the system's knowledge database.

The five basic processing functions performed by each level of the SP hierarchy are illustrated in Figure 33. Sensory processing at a level is typically performed within the context of a task assigned to a BG process within the same RCS_NODE. The current task being planned and executed in the BG selects, from a library of entity and event class frames in long-term memory, a set of expected entities, events, and situations that are relevant to the task. This narrows the search for a match between observed entities and entity classes. The task context also influences the selection of gestalt hypotheses that perform grouping functions, as well as the selection of attention functions that window those regions of the image that are important to the task and therefore need to be processed. Those regions that are unimportant to the task can be masked out and ignored.

At the upper right side of Figure 33, a task goal, priorities, and other parameters are specified by a command to a BG process. This information enables the WM to select a list of entity classes that are important for the task from a library of entity class frames that resides in KD. These are arranged in priority order in a list of entities of attention. This list can be used by another process within WM to compute what the entities on the list are expected (or known) to look like and where in the image (or on the egosphere) they are expected (or known) to appear.

At the bottom right, WM provides estimates and predictions of *what* entities are expected to look like and *where* they are in the world. *What* entities look like is defined by the attributes in the entity class frames in the WM. *What* information provides guidance to the heuristic selection of gestalt hypotheses that will be used to control the grouping of subentities into entities. *Where* important entities can be expected to appear in the image can be computed from the state-variables in the entity class frames in the WM. *Where* information provides guidance to the heuristic processes that define windows of attention to be used to control pointing, tracking, and masking operations.

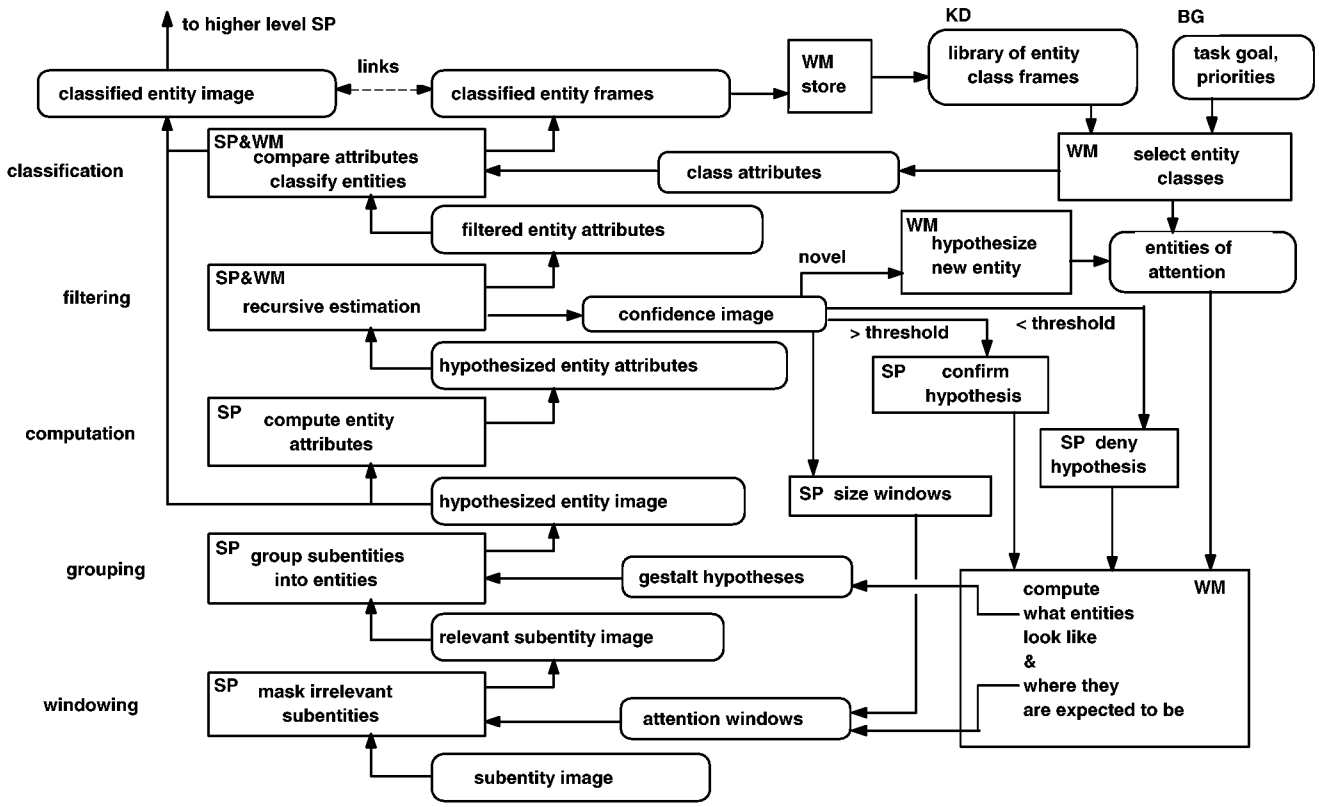


Figure 33. Image processing functions and data flow at a typical level in the SP hierarchy.

At the bottom left, subentity images enter SP to be processed. At the lowest level, the subentity image is simply an array of pixels from the camera. At higher levels, the subentity image might be a list entity image, a surface entity image, or an object entity image. Subentity images are windowed to obtain image regions that are relevant to entities of attention. Subentities that reside within relevant regions are then grouped into hypothesized entities and assigned a group label. The labeled hypothesized entity image then goes two places: (1) it is output to the next higher level, and (2) it is forwarded to the SP computation process where attributes of hypothesized entities are computed. Computed attributes of hypothesized entities are forwarded to a recursive estimation process that filters attribute values and generates a level of confidence in each entity hypothesis. (Note that a confidence factor for each entity produces a confidence image that is registered with the rest of the attribute and entity images). The confidence image is used by three functions: (1) an SP process that confirms or denies the entity hypothesis; (2) an SP process that broadens or narrows the window size; and (3) a WM process that puts novel regions onto the list of entities of attention so that the novel regions can be tracked and identified. The confidence image can also be used to compute confidence values for each of the hypothesized entities.

Finally, at the top left of Figure 33, filtered attributes of confirmed entities are forwarded to a SP/WM classification process where they are compared with attributes of class prototypes. When the attributes of an entity match the attributes of a class prototype, the entity is classified as a member of the class and the class pointer in the entity frame is set to the name (or address) of the class frame. Each pixel in the hypothesized entity image then receives a class label to

form a classified entity image. Classified entities can be stored in the KD, and each pixel in the entity image can then inherit class attributes through its link to the entity frame.

Thus, goals and priorities of the task being planned and executed in the BG hierarchy affects the processing of sensory information in at least three ways:

- ☐ Task knowledge influences the selection of attention functions. Those regions that are likely to contain information important to the task are windowed for processing. Those regions that are unlikely to contain relevant information are masked out and ignored.

- ☐ Task knowledge influences the selection of gestalt hypotheses that perform grouping functions. Those grouping hypotheses that support planned behavior are favored.

- ☐ Task knowledge defines a set of expected entities, events, and situations that are relevant to the task. This generates a set of expected entities and events and narrows the search for a match between observed entities and stored entity classes.

4.5.2 A Typical Processing Node

The relationships and interactions between the BG, WM, KD, SP, and VJ processes in a typical node of the 4D/RCS architecture are shown in Figure 34. The behavior generation (BG) processes contain the Job Assignment (JA), Scheduling (SC), Plan Selector (PS) functions and Executor (EX) subprocesses. The Planner (PL) module overlaps BG, WM, and VJ processes. Planning involves the JA, SC, PS subprocesses, the WM simulator, and VJ plan evaluator. The PS subprocess selects the best plan for execution by the EX subprocesses. The World Modeling (WM) process supports the Knowledge Database (KD that contains both long term and short term symbolic representations and immediate experience images. The WM contains the plan simulator, where the alternatives generated by JA and SC are tested and evaluated, as well as mechanisms for generating predicted images that can be compared with observed images. The Sensory Processing (SP) process contains windowing, grouping, and filtering algorithms for comparing predictions generated by the WM process with observations from sensors. SP also has algorithms for recognizing entities and labeling entity images. The Value Judgment (VJ) process evaluates plans and computes confidence factors based on the variance between observed and predicted entity attributes.

Each node of the 4D/RCS hierarchy closes a control loop. Input from sensors is processed through SP and used by the WM to update the knowledge database (KD). This provides a current best estimate $\underline{X}$ and a predicted state X^* of the world. A control law is applied to this feedback signal arriving at the EX subprocess. The EX subprocess computes the compensation required to minimize the feedback error between the desired state Xd and the predicted state X^* which emerges as a result of the SP/WM filtering process. The predicted state X^* is also used by the JA and SC functions and by the WM plan simulator to perform their respective planning computations. Labeled entity images may also be used to generate labeled maps (not shown in Figure 34) for path planning by the BG process.

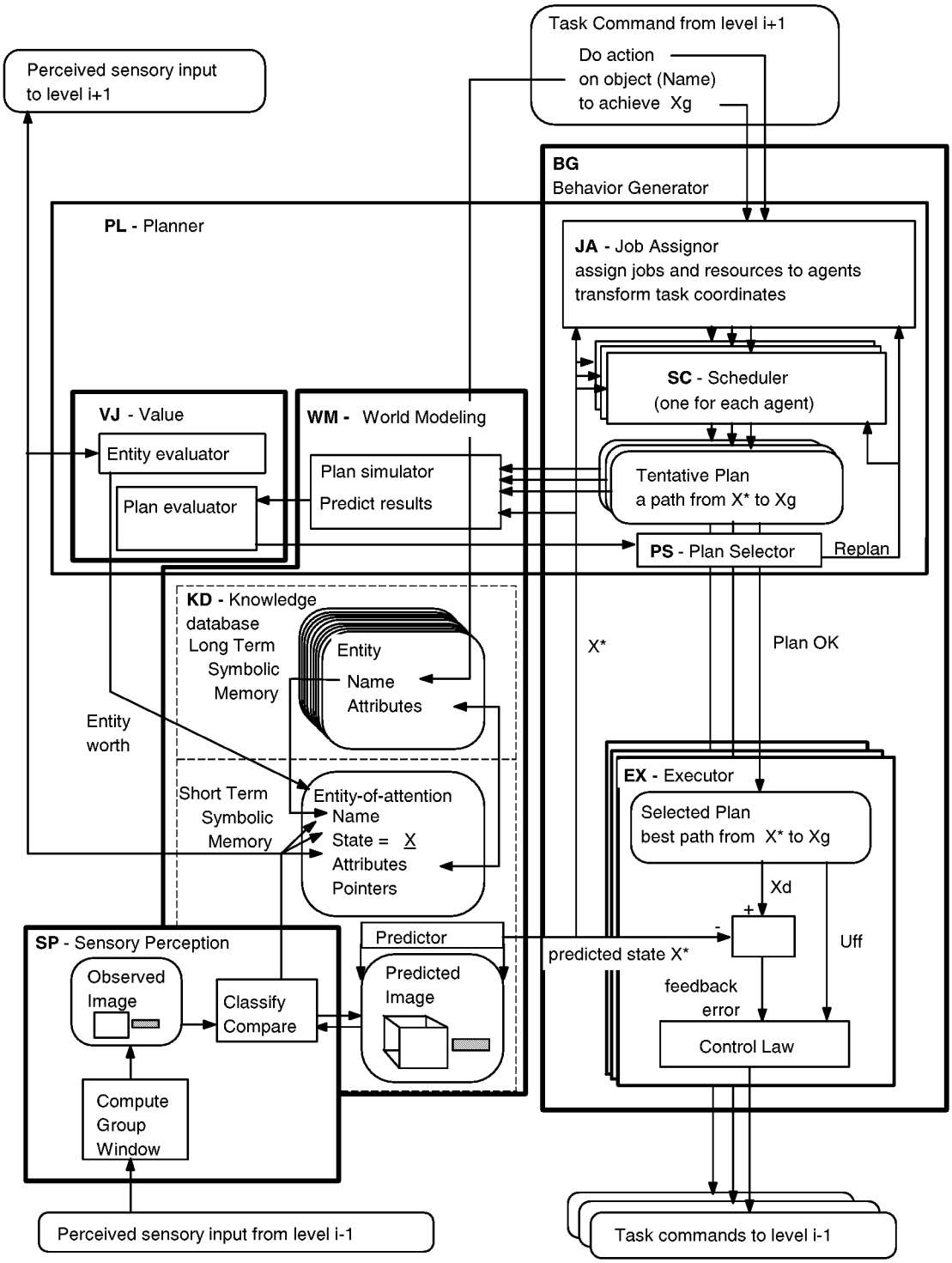


Figure 34. Relationships within a typical node of the 4D/RCS architecture A task command from level i+1 specifies the goal and object of the task. The goal specification selects an entity from long term memory and moves it into short term memory. The Planner generates a plan that leads from the starting state to the goal state. The EX subprocesses execute the plan using predicted state information in the KD short term memory. SP operates on sensory input from level i-1 to update the KD within the node, and sends processed information to level i+1.

Within each node, windowing, grouping, computation, filtering, and recognition take place. Also within each node, planning and control functions are based on the knowledge maintained in the knowledge database, and lower level entities are recognized and grouped into higher level entities. In each node, a model of the world is maintained that enables planning and control functions relevant to that node; and an internal model of the world enables sensory processing to analyze the past and behavior generation to predict the future so as to maximize that node's ability to achieve commanded behavioral goals.

4.5.3 Hypothesize and Test

The SP and WM processes work together to establish and maintain a correspondence between the real world and a model of the world in the knowledge database. SP processes combine observations from sensors and lower level SP processes with predictions based on knowledge already in the KD. Knowledge at time t is a combination of prior knowledge at time t-1 plus observations from sensors and control signals sent to the output at time t. This can be expressed as

$$KD(t) = KD(t\text{-}1) + Observation(t) \tag{4.1}$$

where:

KD(t) is the knowledge represented in the knowledge database at time t

and

$$Observation(t) = Sensory\ input(t) + Control\ output(t) \tag{4.2}$$

The KD is thus a current best estimate of the state of the world based on a-priori information augmented by the entire history of all past observations of sensory input and the entire string of tasks and commands that brought the system to its current state. It is this real-time knowledge in the KD that allows the WM to predict the future and the BG to plan and execute behavior that is intelligent. The ability to predict is crucial to improving the likelihood of achieving behavioral goals amid the uncertainties of a world filled with mostly disinterested entities and acts of nature, but interspersed with sometimes cooperative and often competitive or hostile behavior of other intelligent agents.

Knowledge in the KD provides information for directing attention and masking or windowing input from sensors. At each computational cycle t = k, sensory input is processed in the context of the current task and the current knowledge database. The current model of the world stored in the KD provides the basis for filtering of noisy data and predicting the future state of the world. SP processes compare what is observed by sensors with what is predicted by the WM. Good correlation confirms what is predicted and validates the models and hypotheses upon which the predictions are based. Under these circumstances, BG processes can confidently plan for the future and extrapolate through periods when sensory data is unavailable or noisy. Small variance or correlation offsets may be used to update and correct the information in the

KD and to refine predictive models. Poor correlation, or large variance between what is predicted and observed, causes the system to reconsider its hypotheses and lower its confidence in the models on which its predictions are based.

A simplified illustration of the hypothesize-and-test interaction between the SP and WM processes is shown in Figure 35. The WM process hypothesizes a prediction based on the set of estimated state variables that reside in the knowledge database. WM predictions are compared in the SP processes with sensory observations. Sensory observations come directly from sensors or from lower level SP processes. Correlations between sensory observations and WM predictions indicate the degree to which WM predictions are correct. Differences and correlation offsets indicate the error between the WM prediction and SP observation. If correlation exceeds threshold, the hypothesis is verified, or confirmed. When the hypothesis is verified, focus-of-attention can be narrowed, and residual difference values can be used to update estimated state variables to improve the hypothesis. This is a recursive estimation process that includes predictive filtering and temporal integration. On the other hand, if correlation falls below threshold, the hypothesis is rejected, focus-of-attention is broadened, and a new hypothesis is generated.

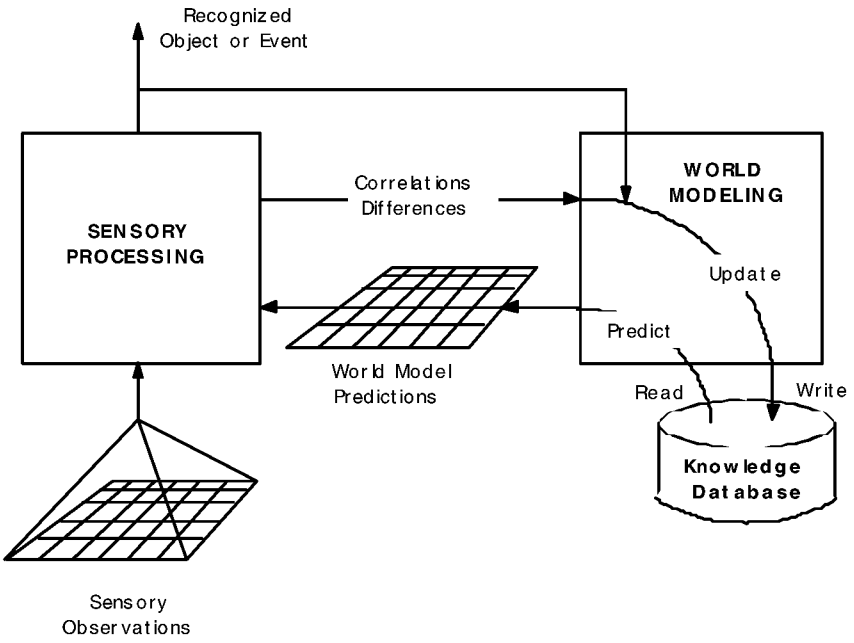


Figure 35. Interaction between sensory processing (SP) and world modeling (WM). SP processes compare sensory observations from sensors or from lower level SP processes with predictions generated by WM processes from knowledge sorted in the knowledge database. Differences are used to update state variables in the KD. Correlations that exceed threshold verify the hypotheses upon which the predictions are based.

4.5.4 Recursive Estimation

Recursive estimation is a mathematical process by which a current best estimate of the state of the world is maintained in the knowledge database. Given this estimated state, the world model can generate predicted sensory input that can be compared with observed sensory input. The variance between observed and predicted sensory data then used to compute a new best estimate or the world state. This can be expressed by the formula

$$\underline{x}(t|t) = x^*(t|t-1) + K(t)(y(t) - y^*(t|t-1)) \tag{4.3}$$

Where:

$\underline{x}(t\|t)$	= estimated state of the world at time t after a measurement at time t
$y(t)$	= observed sensory input at time t
$y^*(t\|t-1)$	= predicted sensory input at time t based on estimated state at time t-1 plus control output at time t-1
$K(t)$	= inverse measurement model and confidence factor

Within each node of the 4D/RCS architecture, interactions between SP observations and WM predictions constitute a recursive estimation loop. The SP processes operate on data from sensors (or from lower level SP processes) to compute observed attributes of entities y(t). WM processes simultaneously generate predicted current attributes $y^*(t|t-1)$ based on the previous best estimate of the state of the world $\underline{x}(t-1|t-1)$ stored in the knowledge database. SP processes compare observed attributes with predicted attributes and compute variance. The WM processes then use the variance $y(t) - y^*(t|t-1)$ to update the knowledge database, producing a new best estimate $\underline{x}(t|t)$. Statistics are kept on the variance to assess the confidence in the model.

Hypothesize and test by recursive estimation is the fundamental paradigm of the 4-D approach pioneered by Dickmanns et al over the past decade [Dickmanns 95, Dickmanns, et al. 94, Dickmanns 92a, Dickmanns 92b, Dickmanns and Graefe 88]. Dickmanns [92a] explains Figure 36 as follows:

At the upper left, the real world is shown by a block; control inputs to the self vehicle may lead to changes in the visual appearance of the world either by changing the viewing direction or through egomotion. The continuous changes of objects and their relative position in the world over time are sensed by CCD-sensor arrays (shown as converging lines to the lower center, symbolizing the 3D to 2D data reduction). They record the incoming light intensity from a certain field of view at a fixed sampling rate. By this imaging process the information flow is discretized in two ways: There is a limited spatial resolution in the image plane determined by the pixel spacing in the camera, and a temporal discretization determined by the scan rate of the camera.

Instead of trying to invert this image sequence for 3D scene understanding, a different approach of analysis through synthesis has been selected, taking advantage of the available recursive estimation scheme from Kalman. From previous experience, generic models of objects in the 3D-world are known in the interpretation process. This comprises both 3D shapes,

recognizable by certain feature aggregations given the aspect conditions, and motion behavior over time. In an initialization phase, starting from a collection of features extracted by low level image processing (lower center left in Figure 36), object hypotheses including the aspect conditions and the motion behavior (transition matrices) in space have to be generated (upper center left in Figure 36). They are installed in an internal 'mental' world representation intended to duplicate the outside real world. This is sometimes called 'world 2', as opposed to the real 'world 1'.

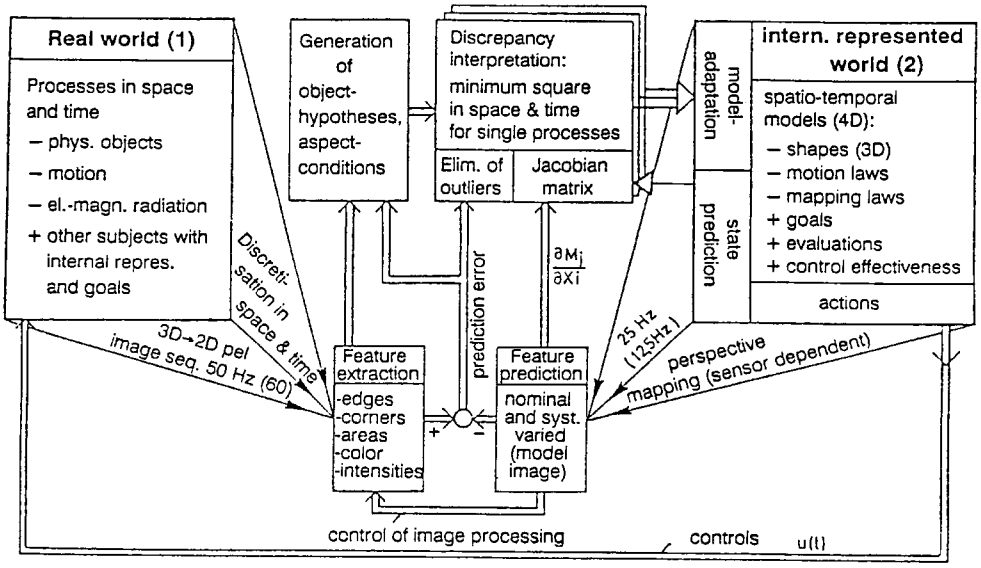


Figure 36. Basic scheme for 4-D image sequence understanding by prediction error minimization [Dickmanns 92a].

Once an aggregation of objects has been instantiated in the world 2, exploiting the dynamical models for those objects allows the prediction of object states for that point in time when the next measurements are going to be taken. By applying the forward perspective projection to those features which will be well visible, using the same mapping conditions as in the CCD sensor, a model image can be generated which should duplicate the measured image if the situation has been understood properly. The situation is thus 'imagined' (right and lower center right in Figure 36). The big advantage of this approach is that due to the internal 4D-model not only the actual situation at the present time but also the Jacobian matrix of the feature positions and orientations with respect to all state component changes can be determined (upper block in center right, lower right corner). This need not necessarily be done by analytical means but maybe achieved by numerical differentiation exploiting the mapping subroutines already implemented for the nominal case. This rich information is used for bypassing the perspective inversion via recursive least squares filtering through feedback of the prediction errors of the features. More details are available in [Dickmanns and Graefe 88].

A more recent version of the 4-D approach is shown in Figure 37. As in Figure 36, the real world in 3-D space and time is mapped through camera optics onto a 2-D array of pixels in a camera. Features are measured in the resulting image and compared with predicted features derived by perspective mapping from predictions and expectations in an internal representation of the real world in 3D space and time (i.e., 4-D). Predicted features are used to mask the incoming image so as to eliminate regions of the image that are not of interest. This is intelligent control of feature extraction.

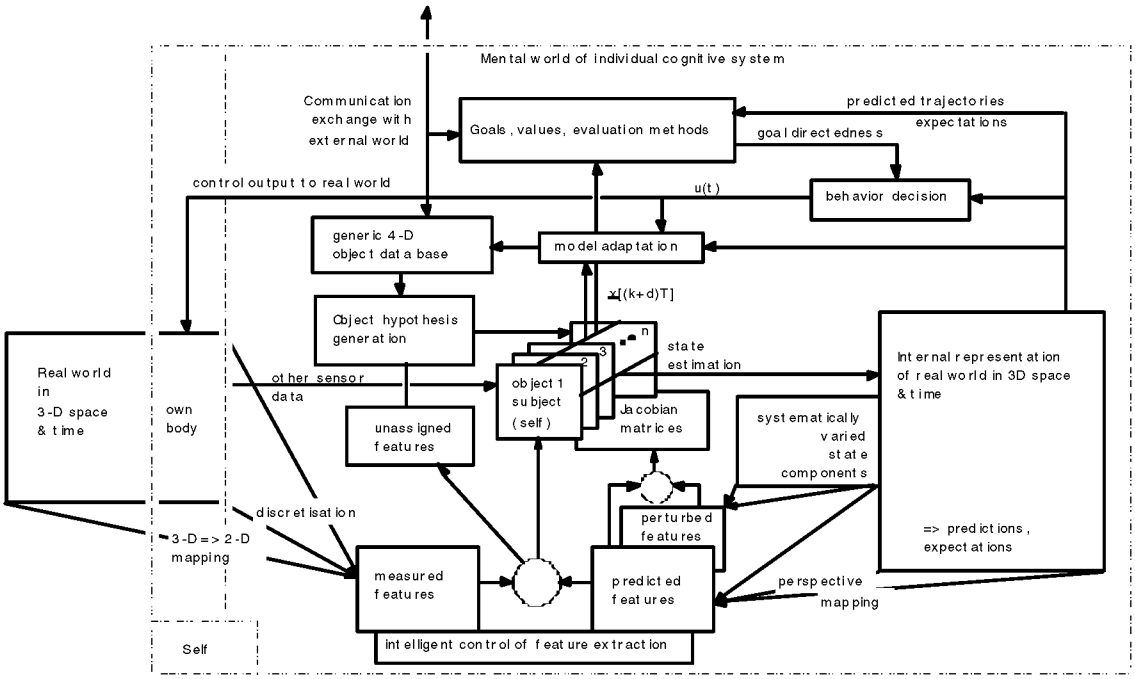


Figure 37. A combination of the 4-D approach with RCS [redrawn from Dickmanns 96].

In those selected regions and for those measured features that correlate with predicted features, correlation offsets are computed and used to update the state vectors of a selected set of objects that are hypothesized to account for the measured features. As shown in Figure 37, there are n objects (starting with object-1, the self object). All objects have state vectors that are represented in a 4-D coordinate frame whose three direction vectors are defined by the local vertical and the tangent vector of the road, with the spatial origin set to the point on the road center-line nearest to the vehicle. The time origin is at t = 0.

Correlation offsets are computed in image coordinates and must be transformed into the 4-D coordinate frame to update the object state vectors. This is accomplished by means of a numerically computed inverse Jacobian. By systematically varying components of the object state vectors, it is possible to compute a Jacobian matrix for each hypothesized object that defines how the correlation offsets are related to changes in the state vector. A pseudo inverse is

then computed and used to update the object state vectors in a recursive estimation loop. This corresponds to the K(t) function in formula (5.4).

The recursive estimation loop can compute an estimate of the state of each object, including its linear and rotational position, orientation, and velocity. A dynamical model of the world embedded in a state-transition matrix then produces predicted states for each of the objects. These predicted states are then transformed through a forward measurement model to produce predicted image features for the next computation in the state estimation loop. The UBM 4-D software closes this loop for up to 16 objects at 20 times per second [Dickmanns 96].

Measured features that do not correlate with predicted features are labeled as "unassigned features" and sent to an object hypothesis generation process which draws upon a generic 4-D object database. The object hypothesis generator may then create a new hypothesized object for the list of objects. Hypothesized objects may be confirmed by good correlation between measured features and predicted features. Confirmed objects may be entered into the generic 4-D database.

Goals input to the generic 4-D object database may also initiate the generation of new hypothesized objects. Goals are compared with the internal representation of the world in a behavior decision process to generate control signals for the self object actuators. Predicted trajectories can be evaluated for purposes of planning. Commanded and predicted trajectories may also be used for model adaptation.

In the UBM 4-D approach, objects are the first level at which recursive estimation is performed. Lower level detection of edges and surfaces are not individually tracked or filtered over time. Objects in the current UBM system are relatively simple, consisting for the most part of combinations of vertical and horizontal edges. However, more complex objects can be incorporated.

Figure 38 is a more complete view of the UBM 4-D approach that shows more detail about how feedback and feed forward control is accomplished and how objects become instantiated in the 4-D world model. On the left, goals and values trigger mission elements consisting of "good" control time histories (i.e., plans or schema) that have been learned from experience. These produce both feed forward control commands and corresponding commanded states that are compared with estimated states to provide error signals for feedback control computation. Feedback compensation is added to feed forward commands and sent to the actuators.

On the right, the 4-D model stores generic classes, dynamical models, and shape models. Dynamical models are initialized by measurement and aggregation of features into hypothesized objects. Dynamical models provide estimates and predictions of object position and aspect conditions. Shape models are used to check for visible features.

Object state predictions are transformed through perspective projection into predicted features that can be compared with measured features to compute prediction error feedback.

This error is applied to disturbance recognition processes that update Kalman filter gains. The error also provides measure of goodness for testing the object hypothesis.

Upward flowing information pathways provide input to a higher level control system that recognizes situations, adapts model parameters, selects goals, triggers behaviors, and evaluates results.

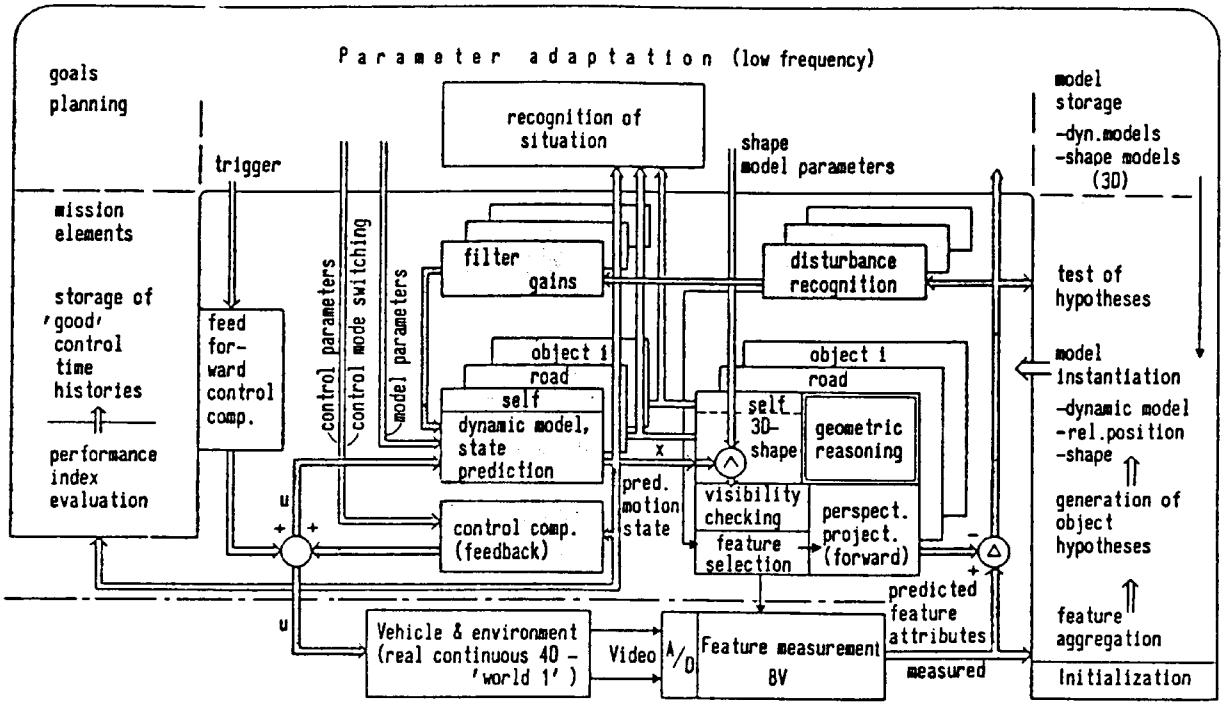


Figure 38. Another view of the 4-D approach [from Dickmanns 96].

Figure 39 is a distillation of the basic 4-D principles shown in Figures 36 through 38 redrawn in a form that shows the interaction with the 4D/RCS BG process. The input to the BG process is a task command from a higher level BG process. The planner module generates a plan to perform the task. This plan is a state graph consisting of a series of planned actions and a series of planned results (or subgoals). The planned actions are feed forward commands that are issued by the BG executor at the appropriate time. The subgoals are desired states Xd that are compared with predicted states X* to generate an error signal that is input to a feedback controller. Output from the feedback controller Ufb is combined with the feed forward commands Uff and sent to the next lower level in the BG hierarchy, or to the actuators (or "virtual actuators" as defined by Albus and Meystel [96]).

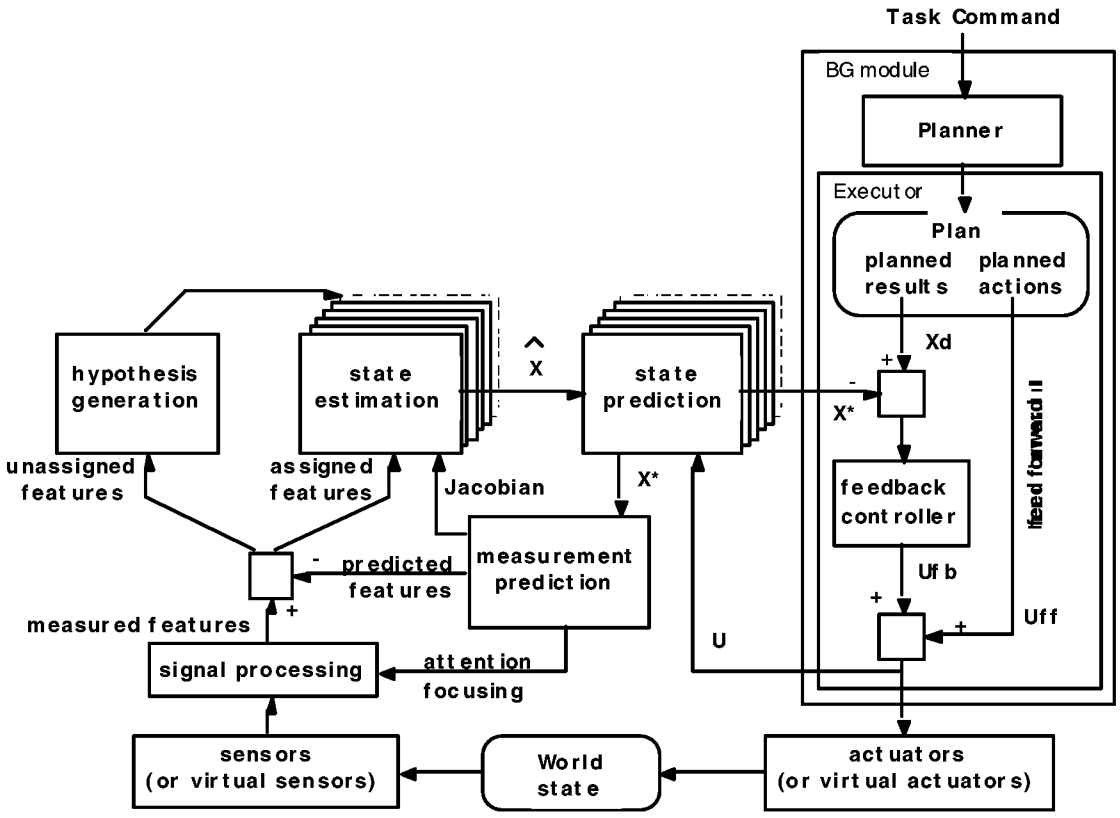


Figure 39. The 4-D recursive estimation concept integrated with a generic RCS BG process.

Figure 40 illustrates in more detail how the 4-D concepts shown in Figures 36 through 39 can be integrated with those of the RCS. At the upper right, a task command input into a BG process causes the job assignor (JA) and scheduler (SC) functions to generate a string of hypothesized actions. These are sent to a dynamic model of the world that generates a string of simulated results. The actions and results are analyzed by cost functions in a VJ (value judgment) process and the evaluations are provided to the plan selector. The hypothesized actions and simulated results constitute a tentative plan. The planner selects the tentative plan with the best evaluation as the plan to be executed. The plan is represented as a state graph in which nodes are simulated results (subgoals, or desired states) and edges are hypothesized (planned) actions. At execution time, the executor issues planned actions to actuators as feed forward commands. These may be commanded camera motion or other commanded actions that affect the image. Predicted states of objects are compared in the executor to subgoals (desired states) from the plan. Differences are used to compute feedback compensation that is added to feed forward commands.

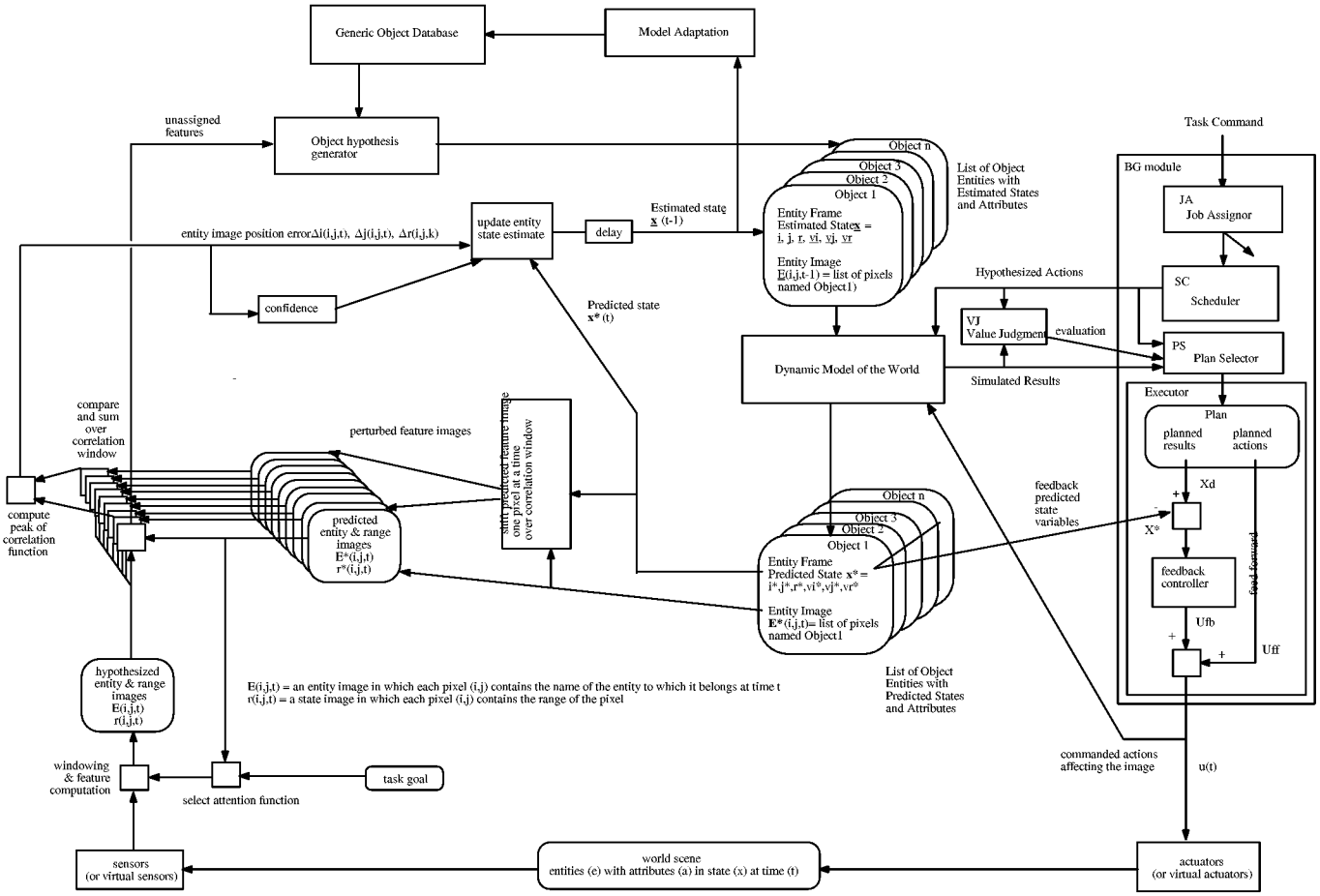


Figure 40. The 4D/RCS for a generic level using camera egosphere coordinates for object state estimation and prediction.

On the lower left, sensors such as TV cameras or a laser range imager produce *sensory input*. These inputs are windowed and features are extracted using knowledge of predicted attributes and states of object entities. The result is a set of measured features such as brightness edges and range values. These measured features are compared with predicted features. A set of perturbed feature images produced by shifting each predicted feature one pixel at a time over the correlation window is compared with the measured feature image and summed over the correlation window. This produces a correlation function. The peak of the correlation function yields the position error between the predicted and measured images. This error is used to update the estimated vector $\underline{\mathbf{x}}$.

The estimated state vector is input to a dynamic model of the world which yields a predicted state vector $\mathbf{x}^*$. Note that in Figure 40, all the entity state variables are expressed in egosphere coordinates and recursive estimation is performed in egosphere coordinates. This eliminates the need for perspective projection and for computation of the Jacobian and the pseudo-inverse that are required in Figures 36 through 39.

4.5.5 Pyramids of Scale

At any level of the 4D/RCS hierarchy, any attribute image may be represented simultaneously at several different scales (or resolutions) in a pyramid of images. The scale or resolution of an image is determined by the size of the spatial and temporal sample collected by each pixel. For any image, a lower resolution image may be derived by subsampling, or by averaging attributes from several higher resolution pixels, to compute an attribute for each lower resolution pixel. This process can be repeated to generate several levels of lower resolution. Each higher level in the pyramid is lower in resolution by a geometric progression that is determined by the ratio of subsampling (i.e., the number of pixels at each level that are averaged at each level to yield a pixel at the next lower resolution.)

In the 4D/RCS, the hierarchy of resolutions provided by pyramid processing is orthogonal to the hierarchy of entity classes and tasks that define levels in the 4D/RCS system. This means that each entity image at each level in the geometrical entity class hierarchy can have its own pyramid of resolution. In fact, each attribute image in Figure 21 can be represented at four different levels of resolution: as a 256x256 pixel image, a 128x128 pixel image, a 64x64 pixel image, and a 32x32 pixel image.

The utility of pyramids of scale is primarily a reduction in the computational expense of correlating two images. To compute a correlation value between a predicted and observed image segment, the value of each pixel attribute in the predicted image segment must be multiplied by the value of the corresponding pixel attribute in the observed image segment and the results summed over the correlation window. To compute the correlation function, a correlation value must be computed for every possible offset between predicted and observed image segments. The number of calculations required to find the correlation function therefore increases as the product of the number of possible offsets along each degree of freedom multiplied by the size of the correlation window, all raised to the power of the number of degrees of freedom. This computational burden can be minimized by reducing the resolution, making the correlation window small, or reducing the number of degrees of freedom.

Pyramid processing minimizes both the resolution and the size of the correlation window at each level of the pyramid, yet maintains high resolution in the resulting correlation function by successive approximations at several levels of scale. In pyramid processing, correlation is performed first at the lowest resolution of the pyramid, and for only zero and plus or minus one pixel offset in x and y directions (a total of 9 offsets). The resulting low resolution correlation function can be used to make a course correction of the correlation offset. Correlation can then be performed at a higher resolution scale, again for only zero and plus or minus one pixel offset. A second more refined correction can then be made. This process can be repeated at each successively finer scale of the pyramid, until the finest scale is reached. This is a form of binary search for the correlation offset peak. By this procedure, the number of computations required to compute the correlation function is significantly reduced. Correlation by pyramid processing requires only the logarithm[8] of the number of computations required by correlation at the finest scale.

The peak of the correlation function indicates how much the predicted and observed image segments are shifted relative to each other along each axis of freedom. The amount of shift divided by the temporal separation between the two images provides a measure of the image flow rate along each axis of freedom, i.e.:

$$dx/dt = \Delta x / \Delta t$$
$$dy/dt = \Delta y / \Delta t$$

where dx/dt is the image flow rate along the x-axis
dy/dt is the image flow rate along the y-axis
Δx = correlation peak offset along the x-axis
Δy = correlation peak offset along the y-axis
Δt = temporal separation between images

Pyramid processing techniques have been extensively studied at the David Sarnoff Research Laboratory by Dr. Peter Burt and his associates. The Sarnoff team has developed algorithms and special purpose computing hardware to implement pyramid processing for area based correlation in real time. The hardware and software techniques developed by the Sarnoff team vastly improves the speed and efficiency of the computation of correlation and difference values over what can be achieved by conventional methods. The pyramid processing algorithms also provide means for widening and narrowing the focus of attention, and for electronically stabilizing images [Burt and Adelson 83, Burt 88, Burt et al. 89, Burt and van der Val 90].

4.5.6 Levels of Sensory Processing

At each level in the SP hierarchy there are five basic processing functions: windowing, grouping, computation, filtering, and recognition. An example of a how these five functions can be implemented at six different levels of the 4D/RCS hierarchy is given below.

[8] The base of the logarithm is the ratio of resolutions at successive levels of the pyramid.

Level 1 Regions: Point Entities and Events (i.e., Pixels and Samples)

The input to level 1 of the SP hierarchy is generated by sensors that detect energy derived from entities and events in the environment. External entities may include actuators, tools, materials, and objects. External events may include shots, explosions, movement of vehicles, passage of lines, arrival-at or departure-from locations, completion of tasks, and accomplishment of goals. Objects and events may interact in relationships and dynamic situations in the world.

Sensors measure phenomena that are caused by states, attributes, or conditions of entities and events in the world. For example, encoders measure position, tachometers measure velocity, accelerometers measure acceleration. Other types of sensors may measure force, torque, temperature, or acoustic energy. Each photodetector in a CCD camera measures the visible radiation imaged on it from a pixel sized region of the world. Pixels sample the spatial dimension. The camera frame rate samples the temporal dimension.

Level 1 Windowing

At level 1, windowing defines the region of space that is sampled by each sensor and the duration over which a signal from each sensor is integrated. Windowing may also define how often each sensor is sampled. For acoustic or microwave sensors, windowing may be performed by pointing directional microphones or antennas at high priority targets. For position, orientation, velocity, acceleration, and force sensors, windowing may consist of selecting those signals that are relevant to the current servo task command for each actuator. For vision sensors, windowing includes pointing and focusing each camera's photodetector array on the region of the egosphere that is most worthy of attention.

For example, on Demo III vehicles, there are several cameras of different types with different resolutions, fields of view, and frame rates. Based on the assigned mission, an attention subsystem will focus the highest resolution cameras on the most important region in the image while using lower resolution cameras to scan for other important regions. The attention system may track the highest priority entity of attention, or saccade quickly from one high priority entity to the next as the priority list is updated.

Level 1 Grouping

At level 1, grouping simply means that each sensor integrates, or groups, all the energy impinging upon it during a sample interval. For example, each photodetector in a CCD camera integrates the incoming visible radiation over the spatial region occupied by that photodetector and over the temporal interval of an exposure time period.

Level 1 Measurement and Computation

A number of attributes can be directly measured for each pixel. In a black and white TV camera, the intensity (I) of radiation at each pixel is measured. In a color TV camera, the intensity of red, blue, and green light (r, g, b) at each pixel is measured. In a laser range imager

(LADAR), or an imaging radar, a value of range (r) is measured at each pixel. For a forward looking infrared (FLIR), the temperature at each pixel is measured. A pair of cameras with overlapping fields of view may provide stereo image pairs.

Additional attributes for each pixel can be computed. For example, the x- and y-intensity gradients (dI/dx, dI/dy) may be computed at each pixel by calculating the difference in intensity between adjacent pixels in the x- and y-directions, or by applying a gradient operator such as a Sobel operator on a neighborhood about a pixel. The temporal gradient (dI/dt) can be computed by subtracting the intensity of a pixel at time t-1 from the intensity at time t, or by applying a temporal gradient operator. Differences in range at adjacent pixels can be used to compute spatial range gradients (dr/dx, dr/dy) and surface roughness, or texture (tx). Spatial range gradients can also be computed from shading (dI/dx, dI/dy) when there is knowledge of the incidence of illumination. Temporal differences in range at a point can yield range rate (dr/dt).

For a pair of stereo images, disparity can be computed at each pixel by a number of algorithms. When combined with camera vergence and knowledge of camera spacing, disparity can be used to compute range for each pixel at every point in time.

In cases where the camera image is stabilized (or camera rotation is precisely known), spatial and temporal gradients can be combined with knowledge of camera motion derived from inertial and speedometer measurements to compute image flow vectors (dx/dt, dy/dt) at each pixel (except where the spatial gradient is zero. In this case, the flow rate sometimes can be inferred from the flow of surrounding pixels). For stationary objects, time to contact (ttc) can be computed from image flow. Under certain conditions, range at each pixel can also be computed from image flow. [Albus and Hong90]

Level 1 Filtering

Each attribute image can be filtered in order to reduce or eliminate noise and to improve signal to noise ratio. Figure 41 is an example of how recursive estimation might be used to filter attribute images at level 1. In this example, each observed (i.e., measured or computed) attribute image is compared with a predicted attribute image. The prediction process uses an estimate of the image flow rate at each pixel to predict where a pixel attribute observed in an image at time t will occur in the next image at t+1. The estimated flow rate at each pixel combines information from two sources:

1. Estimated image flow rates ($\underline{dx}/\underline{dt}$, $\underline{dy}/\underline{dt}$) and range rate ($\underline{dr}/\underline{dt}$) that describe the current motion of entities in the image.
2. Commanded camera motions and other actions that affect motion of entities in the image.

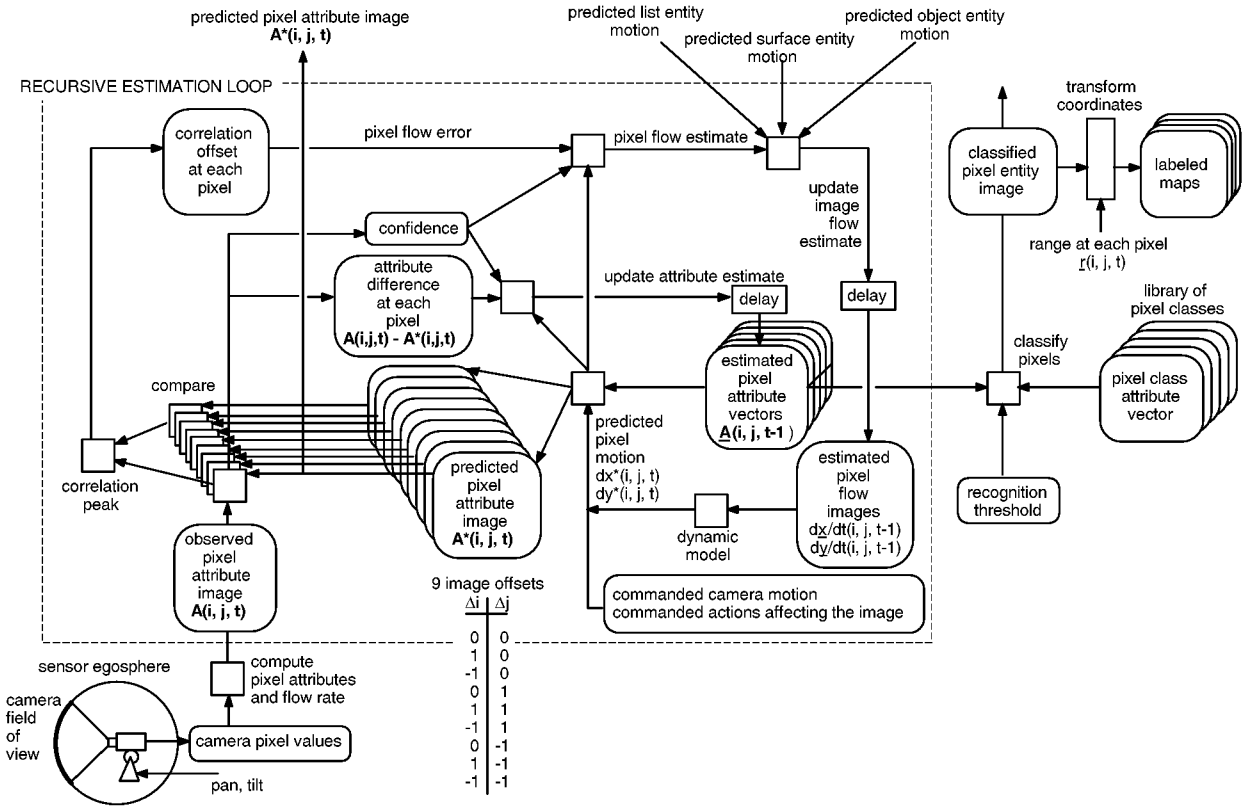


Figure 41. Image processing at level 1. Pixel signals from cameras are used to compute observed pixel attribute images. These are compared with predicted pixel attribute images and the difference is used to update estimated pixel attribute images. Perturbed attribute images are used to compute a correlation function over a set of mage offsets. The peak of the correlation function is used to compute errors in the estimated pixel flow. This is combined with higher level estimates of entity motions to estimate pixel flow at each pixel. Predicted pixel attribute vectors are compared with pixel class attribute vectors to classify pixels and generate a labeled pixel class image.

The computation of image flow at a single pixel at a single point in time is notoriously unreliable and noisy. However, filtering over an interval in time by recursive estimation has been shown to provide considerable improvement in signal-to-noise ratio. Integration over a group of points comprising an entity further improves the signal-to-noise. The computation of image flow from a combination of entities at several different levels, each of which is recursively estimated, can provide a robust estimate of image flow at each pixel. This is illustrated in Figure 41 where the estimated flow rates dx/dt and dy/dt are computed from a combination of estimates, including:

1. From the correlation offset between the predicted attribute image and the observed attribute image at each pixel.
2. From the list entity flow estimate for the list entity to which the pixel belongs.
3. From the surface entity flow estimate for the surface entity to which the pixel belongs.
4. From the object entity flow estimate for the object entity to which the pixel belongs.

In equation form this can be expressed as:

$$\underline{dx/dt}(x,y,t\text{-}1) = c1 * \underline{dx/dt}(pixel) + c2 * \underline{dx/dt}(list) + c3 * \underline{dx/dt}(surface) + c4 * \underline{dx/dt}(object)$$
$$\underline{dy/dt}(x,y,t\text{-}1) = c1 * \underline{dy/dt}(pixel) + c2 * \underline{dy/dt}(list) + c3 * \underline{dy/dt}(surface) + c4 * \underline{dy/dt}(object)$$

Where:

c1 = confidence in the pixel level flow computation for the pixel at x, y

c2 = confidence in the list level flow computation for the pixel at x, y

c3 = confidence in the surface level flow computation for the pixel at x, y

c4 = confidence in the object level flow computation for the pixel at x, y

And

$$c1 + c2 + c3 + c4 = 1$$

This is illustrated in Figure 41. In the outer loop of Figure 41, correlation offset at each pixel generates a measure of pixel flow error. This is added to the predicted flow rate to generate an estimate of pixel flow rate. This is then combined with higher level estimates of list entity flow, surface entity flow, and object entity flow to compute a best estimate of flow at each pixel.

Once a reliable estimate of image flow is known, the attribute of a pixel at position (x,y) at time t-1 can be predicted to appear at position (x+dx, y+dy) at time t. This can be expressed as:

$$I^*(x+dx, y+dy, t) = \underline{I}(x, y, t-1)$$

In Figure 41, a predicted intensity image I*(x,y,t) is generated from an estimated intensity image $\underline{I}(x,y,t-1)$ by computing the expected image motion for each pixel attribute based on image flow estimates $\underline{dx}/\underline{dt}(x,y,t-1)$ and $\underline{dy}/\underline{dt}(x,y,t-1)$ at each pixel, plus the effect of control signals sent to the camera pan/tilt unit, plus other control actions effecting the image.

Comparison between predicted and observed images produces correlation and difference values. The attribute difference between observed image I(x, y, t) and predicted (with zero offset) I*(x, y, t) is used to update the attribute estimate. Correlation of the observed image with a set of predicted images with different offsets produces a correlation function. The offset values that produce the maximum correlation value indicate the amount of position error (Δx and Δy) between the observed and predicted attribute images. These position errors divided by the sample period yield the pixel flow error.

The combination of image flow estimates illustrated in Figure 41 suggests how recursive estimation can provide a means for combining information about a single estimated pixel attribute from several sources. Each level in the hierarchy produces an estimate of image flow for each pixel. The higher levels estimate the flow rate of entities consisting of many pixels. Each higher level computes flow estimates for entities with more pixels. Thus, combining flow estimates from multiple levels produces good estimates of individual pixel flow rates at the lowest level

Level 1 Classification

The final SP function performed at each level is classification. At level 1, this may be accomplished by comparing estimated pixel attributes with attributes of a library of pixel (geometric point) class prototypes. The set of attributes for each pixel define a pixel attribute

vector. Geometric point classes might include: brightness-discontinuity, color-discontinuity, range-discontinuity, range-gradient-discontinuity, and surface-continuity. Geometric point class prototype attributes may include discontinuity size, shape, position, velocity, and orientation. When the inner product of the estimated pixel attribute vector and the pixel class prototype attribute vector exceeds threshold, the pixel is classified as belonging to the class. A single pixel may belong to more than one geometric point class. For example, a single pixel may belong to a brightness-discontinuity class, a range-discontinuity class, and a color-discontinuity class.

Upon classification, each pixel acquires a pointer to the pixel class to which it belongs. The classification process thus generates a pixel entity image where each pixel in each entity image has pointers to the pixel classes to which the pixel belongs.

Summary of level 1:

1. A portion of the egosphere is windowed onto each sensor.
2. The energy falling on each sensor is integrated over a sample interval in space and time.
3. Observed values are computed for a set of attributes for each pixel.
4. Pixel attributes are filtered (possibly by recursive estimation). Estimated and predicted values are computed for each attribute of each pixel by a recursive estimation filter process.
5. Each pixel is classified as a member of a geometrical point entity class, and a pixel entity image is formed in which each pixel has a pointer (or pointers) to the class (or classes) to which it belongs.

Note that in Figure 41 there are two recursive estimation loops. The first loop is for estimating state (i.e., position and velocity) at each pixel. The second loop is for estimating attributes such as size, orientation, and class. The position of predicted attribute images may be systematically perturbed in order to find the peak of the correlation function and compute the correlation offset at each pixel. This can then update the flow and range-rate estimates for each pixel. The pixel flow and range-rate estimates also incorporate higher level entity motion estimates for list, surface, and object entities to which that pixel belongs. For estimating pixel attributes other than state, the error between observed and predicted attributes can be used directly to update estimated attribute values.

The two recursive estimation loops shown in Figure 41 are analogous to the "WHAT" and "WHERE" channels that are known to exist in the visual processing hierarchy in primate brains.

In the 4D/RCS:

1. The "WHERE" channel computes estimated state (e.g., position and velocity) of entities in the image. Estimated state is then used to predict where to expect entities to appear next in the image.

2. The "WHAT" channel computes attributes that describe distinguishing characteristics of entities in the image. These include brightness, color, size, shape, texture, sound, taste, feel, or smell.

Both "WHERE" and "WHAT" attributes can be used to identify, recognize, classify, or name entities in the image, and by projection, in the external world. Both channels also provide information that is useful in windowing, grouping, and segmentation. The "WHERE" information may be used to group subentities into entities (or regions) based on state, (i.e., how subentities are connected, or how they move together in the image.) The "WHAT" information may be used to group subentities based on similarity of attributes or class.

Level 2 Regions: List entities and events

At level 2 of the SP hierarchy, groups of a few (on average about ten) pixels with attributes that correspond to a list entity class can be grouped, processed, and analyzed as level 2 regions, or list entities. For example, a group of pixels with contiguous brightness or color gradients might be grouped, processed, and analyzed as a list entity such as an edge, vertex, or surface patch. Also at level 2, temporal strings or patterns of attributes from a single sensor can be grouped, processed, and analyzed as level 2 events. For example, a temporal signal, or sequential pattern of intensity attributes from an acoustic sensor might be grouped as a tone, a frequency, or a phoneme.

Level 2 Windowing

Windowing consists of placing windows around regions in the image that are designated as worthy of attention. At level 2, the selection of which regions to so designate may depend on the goal and priorities of the current level 2 task. The selection of a window may also depend on the detection of noteworthy attributes of pixels or higher level entities. The shape of each window is determined from the shape of recognized list entities. The size of each window is determined by the confidence factor computed by the level 2 recursive filtering process. If the estimated entity confidence value is high, each window will be narrowed to only slightly larger than the set of pixels in the corresponding list entity region. If the estimated entity confidence value is low, the windows will be significantly larger than the level 2 list entity regions. Until there exists a set of recognized list entities, level 2 windows are set wide open.

Level 2 Grouping

At level 2, grouping is a process by which neighboring pixels of the same class with similar attributes can be grouped to form higher level entities. Information about each pixel's class is derived from the classified pixel entity image. Information about each pixel's orientation and magnitude is derived from pixel attribute images. Grouping is performed by a heuristic algorithm based on gestalt principles such as contiguity, similarity, proximity, pattern continuity, or symmetry. The choice of which gestalt heuristic to use for grouping may depend on an attention function that is determined by the goal of the current task, and on the confidence factor developed by the recursive estimation process at level 2. Grouping causes an image to be segmented, or partitioned, into regions (or sets of pixels) that correspond to higher order entities.

For example, at level 2 contiguous pixels in the same geometric entity class (i.e., with similar attributes of range, orientation, and flow rate) can be grouped into level 2 regions. Of course, any grouping is a hypothesis that the pixels in the group all lie on the same entity in the world. The grouping process thus generates a hypothesized list entity image in which each group is assigned a label that is a pointer to a list entity frame. A grouping hypothesis can be tested by a recursive estimation algorithm applied to each of the hypothesized entities. If the recursive estimation process is successful in predicting the observed behavior of a hypothesized entity in the image, then the grouping hypothesis for that entity is confirmed. On the other hand, if the recursive estimation process is not successful in predicting the behavior of the hypothesized entity, the grouping hypothesis will be rejected, and another grouping hypothesis must be selected.

In practice, a number of grouping hypotheses may be tested in parallel if the computational resources are available. In this case, the grouping hypothesis with the greatest success in predicting entity behavior will be selected. In either case, the measure of success in predicting behavior is the confidence factor computed by the recursive estimation algorithm for the estimated entity or event.

The result of the grouping process is that each pixel in the group has a pointer set to the name (or location) of a geometric list entity frame with slots for attributes of the list entity. A back-pointer in each list entity frame also is set to the list entity image as illustrated in Figure 22.

Level 2 Computation

Each of the regions defined by the grouping hypotheses have attributes that can be computed. Entity attributes differ from pixel attributes in that they are integrated over the entire group of pixels comprising the entity. For example, edge entities have attributes such as length, curvature, orientation, position and motion of the edge center of gravity. Surface-patch entities have attributes such as area, texture, average range, average surface gradient, average color, position and motion of the center of gravity. For each hypothesized level 2 grouping, computed entity attributes fill slots in an observed list entity frame.

Level 2 Filtering

At level 2, a recursive estimation process compares observed list entity attributes with predicted list entity attributes that are generated from estimated list entity attributes. Prediction is a function of information from two sources: (1) estimated dynamic attributes such as image flow rates ($\underline{dx}/\underline{dt}$, $\underline{dy}/\underline{dt}$) and range rate ($\underline{dr}/\underline{dt}$) that describe the motion of entities in the image; and (2) commanded actions that affect motion of entities in the image.

Comparison of observed entity attributes with predicted entity attributes produces correlation and difference values. Difference values are used to update the estimated entity attribute values. Estimates of image flow rates of entities at level 2 are generated from a combination of correlation offsets generated at level 2, plus estimates of flow attributes from higher levels. A confidence level for each estimated entity attribute value is computed as a

function of the correlation and difference values. If predictions based on estimated entity attributes successfully track the behavior of observed entity attributes, the confidence level rises. When the confidence level of the recursive estimation filter rises above threshold, the hypothesized list entity grouping is confirmed. If the confidence level of the recursive estimation filter falls below threshold, the hypothesized level 2 grouping is rejected, and another grouping hypothesis must be selected.

Level 2 Classification

The classification process compares the attributes of each estimated list entity with attributes of *geometric* list entity class prototypes such as range-edges, brightness-edges, color-edges, surface-slope-discontinuity-edges, surface-patches, or vertices of various kinds. Another classification process may compare the attributes of each estimated list entity with attributes of *generic* list entity class prototypes such as road edges, building edges, tree trunk edges, patches of ground, the near edge of a ditch, the far edge of a ditch, the crest of a hill.

Upon classification, the list entity frame and all the pixels in the list entity image have a pointer set to the name (or names) of the list entity class (or classes) to which the frame and all of its pixels belong. Each list entity frame also has a back-pointer to the list entity image.

Summary of Level 2:

(1) Regions of the image containing list entities of attention are windowed.
(2) Pixels with similar attributes are tentatively grouped into level 2 regions, or hypothesized list entities.
(3) The attributes of each hypothesized list entity are computed.
(4) The attributes of each hypothesized list entity are filtered by recursive estimation, and each grouping hypothesis is either confirmed or rejected.
(5) The attributes of each confirmed list entity are compared with the attributes of a set of list entity class prototypes, and those that match are assigned to list entity classes. Object entity frames are given pointers to the class to which they belong.

Figure 42 summarizes the set of operations and data performed by vision SP and WM processes at level 2 (and higher). At level 2, the input is the labeled pixel image from level 1. At the lower left of Figure 42, labeled pixels are windowed by a goal directed attention function that masks out pixels that are irrelevant to the goal. A gestalt hypothesis then is used to group labeled pixels into level 2 regions (i.e., list entities such as edges, vertices, and surface patches.) The resulting hypothesized list entity image can then be used to compute attributes and states of list entities. For edge entities, attributes may include length and curvature. For vertices, attributes may include type (e.g., T, Y, V, or line-end vertices). List entity states may include position (x,y,r), orientation (θ), and velocity (vx, vy, vr, vθ). For surface patches, attributes may include area, texture, color, shape, and surface orientation. State includes position and velocity.

Hypothesized list entities can be correlated with a predicted list entity image computed from a previously estimated list entity image. Correlation offsets between each list entity in the hypothesized and predicted image are used to update the estimated motion for each list entity.

Perturbations of predicted entity images produce a correlation function for each list entity from which the correlation offset error for each list entity can be computed. Correlation offset errors are combined with predicted flow rates for each entity to produce list entity flow estimates. These, combined with surface and object motion predictions, provide updated estimates of list entity motion. Estimated motion, combined in a dynamic model with commanded actions, produce predicted list entity motion. This, applied to the estimated list entity image, produces the predicted list entity image.

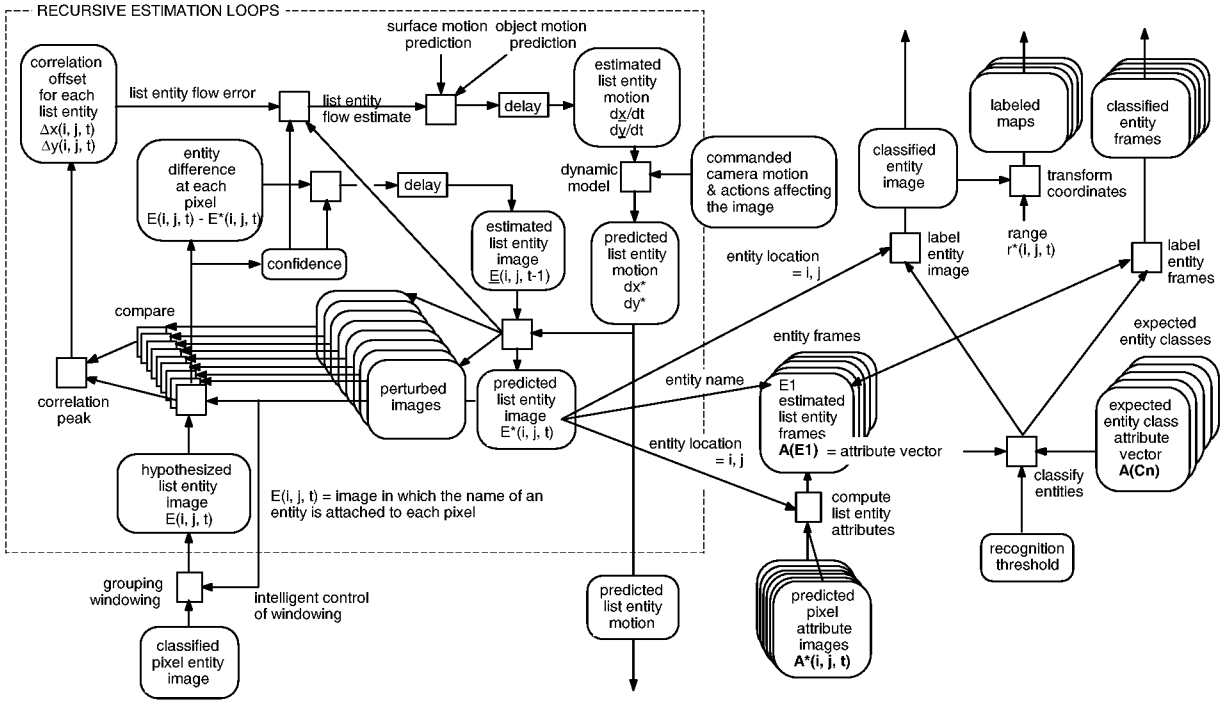


Figure 42. Image processing at level 2 and above. Labeled pixels are windowed and grouped into level 2 regions to form a hypothesized list entity image. This is compared with a predicted list entity image generated by recursive estimation and prediction. Differences are used to update the estimated list entity image. Perturbed entity images are used to compute a correlation function over a set of image offsets. The peak of the correlation function is used to compute errors in the estimated flow for the list entities in the image. This is combined with higher level estimates of entity motions to estimate list entity motion. The predicted list entity image is combined with predicted pixel attribute images to generate predicted list entity frames with list entity attribute and list entity state vectors. The predicted state vector can be used as feedback in the BG module. The predicted attribute vector is compared with expected list entity classes to classify entities. The result is a set of recognized entity frames and a labeled list entity image.

Pointers in the predicted list entity image are used to address pixels in predicted pixel attribute images so as to compute attributes for list entity frames. The attributes of estimated list entity frames are then compared with the attributes of expected list entity class prototypes. If the dot product between the estimated and expected entity class attribute vectors rises above recognition threshold, the estimated level 2 regions are recognized as members of generic or specific list entity classes. The names of the classes are attached as labels to the predicted list entity image regions to produce a labeled list entity image. Class names are also attached to estimated entity frames as a sign of class membership. The result is that class names are

assigned to regions in the entity image and class names are assigned to entity frames that are passed upward to the next level of processing. Predicted entity motion is passed downward to update pixel level motion estimates.

Recursive estimation can be performed on entity attributes and states at all levels. The confidence factor that is generated by the recursive estimation process for each hypothesized entity can be used to confirm or deny the gestalt grouping hypothesis. If the confidence factor rises above a recognition threshold, the hypothesis is confirmed. If confidence falls below a doubt threshold, the gestalt hypothesis is denied and a new grouping hypothesis must be selected.

The recursive loops illustrated in Figure 42 are repeated at level 3 to produce labeled surface and boundary entity images linked to classified surface and boundary entity frames. They are repeated again at level 4 to produce labeled object entity images linked to classified object entity frames. They are repeated again at level 5 to produce labeled group (section) entity images linked to classified group (section) entity frames.

Because each pixel carries an estimated range attribute, labeled entity images can be transformed into world coordinates to produce topographic maps with labeled regions and icons at all levels in KD. Such maps can be used by the behavior generating processes at the corresponding levels for path planning. At each level of the WM and KD that service the BG hierarchy, maps of different range and resolution are used for different planning horizons and different levels of detail.

At each level, expected entities are drawn out of long term memory when referenced by task goals. Unexpected entities are discovered by applying gestalt grouping hypotheses to unassigned subentities with common attributes. Unexpected entities that have significance to behavioral goals are noted and entered into the world model as hypothesized geometric entities. Recursive filtering confirms or rejects the entity hypothesis, and attributes of the unexpected but confirmed entity can be compared with attributes of known entity classes. If the unexpected entity is classified, it inherits the attributes of the entity class to which it belongs. If it is not classified, a new class is established so that it can be recognized the next time it is encountered. Whether or not the unexpected entity is classified as belonging to a known class, it becomes a part of the knowledge database where it may influence behavior generation in a number of ways including the selection of future tasks. For example, the behavior generation system may initiate a task to inspect, pursue, or avoid the unexpected entity.

Level 3 Regions: Surface Entities and Events

At level 3 of the SP hierarchy, groups of list entities with geometric attributes that correspond to a surface entity class can be grouped, processed, and analyzed as level 3 regions called surface entities. For example, a set of contiguous surface patch list entities with similar range and motion might be grouped, processed, and analyzed as a surface entity. A set of edge and vertex entities that are contiguous along their orientation might be grouped, processed, and analyzed as a surface boundary entity. Temporal strings of level 2 events with attributes that correspond to level 3 events can be grouped into level 3 events such as sonograms, or 2-dimensional surfaces in frequency and time. For example, a temporal string of phonemes might

be grouped as a word, or a temporal string of outputs from a set of frequency filters as a note or phrase in an acoustic signature.

Level 3 Windowing

At level 3, windowing consists of placing windows around regions in the labeled list entity image that are designated as worthy of attention. The selection of which regions to window is made by an attention function that depends on the goal and priorities of the current level 3 task, or on the detection of noteworthy attributes of estimated surface entities, or on inclusion in a higher level entity of attention. The shape of the windows is determined by the class of regions in the surface entity image. The size of the windows is determined by the confidence factor associated with the degree of match between the predicted surface entities and the observed surface entities.

Level 3 Grouping

At level 3, recognized list entities in the same class are grouped into level 3 regions called surface entities based on gestalt properties such as contiguity, similarity, proximity, pattern continuity, and symmetry. Surface patches in the same class that are contiguous at their edges and have similar range, velocity, average surface orientation, and average color may be grouped into surface entities. Edges and vertices in the same class that are contiguous along their orientation with similar range and velocity may be grouped into boundary entities. Grouping is a hypothesis that all the pixels in the group have the common property of imaging the same physical surface or boundary in the real world. This grouping hypothesis is confirmed or rejected by the level 3 filtering function.

Level 3 Computation

Observed level 3 regions have attributes such as: area, shape, position and motion of the center-of-gravity, range, orientation, surface texture, and color. Surface boundary entities have attributes such as boundary type, length, shape, orientation, position and motion of the center of gravity, and rotation. Boundary types may include intensity, range, texture, color, and slope boundaries. For each level 3 region, computed entity attributes fill slots in an observed surface entity frame.

Level 3 Filtering

A recursive estimation process compares observed surface entity attributes with predicted surface entity attributes generated from estimated surface entity attributes, planned results, and predicted attributes of parent entities. Comparison produces correlation and difference values. Difference values are used to update the estimated surface entity attributes. Correlation values are used to update estimates of entity motion. A confidence level associated with each estimated entity confirms or rejects the surface entity grouping hypothesis that created it.

Level 3 Classification

The classification process establishes a match between estimated surface entities and a generic or specific class of surface entity prototypes in the world model knowledge database. Typical generic surface entity classes are the surface of the ground, tree foliage, tree trunks, sides of buildings, roofs of buildings, road lanes, fences, or lake surface. As a result of classification, a labeled surface entity image is formed in which each pixel has a pointer to (or the name of) the surface entity class to which it belongs.

Summary of Level 3 Sensory Processing

(1) Portions of the image containing surface entities of attention are windowed.
(2) List entities with similar attributes are tentatively grouped into level 3 regions, or surface entities.
(3) The attribute values of each geometric surface and boundary entity are computed.
(4) Attributes of each hypothesized surface entity are estimated by recursive estimation and each grouping hypotheses is either confirmed or rejected.
(5) The attributes of each confirmed surface entity are compared with the attributes of a set of surface entity class prototypes, and those that match are assigned to section entity classes. Surface entity frames are given pointers to the class to which they belong.

Level 4 Regions: Object Entities and Events

At level 4 of the SP hierarchy, groups of level 3 entities with attributes such as range, motion, orientation, color, and texture that correspond to an object entity class can be grouped, processed, and analyzed as level 4 regions, or object entities. For example, a group of surfaces with coincident boundaries and similar or smoothly varying range and velocity attributes might be grouped into an object such as a building, a vehicle, or a tree. Attributes of surface entities comprising each object entity are combined into object entity attributes. Also at level 4, temporal strings of level 3 events such as words might be grouped into a sentence, or acoustic signatures might be grouped into a level 4 event.

Level 4 Windowing

At level 4, windowing consists of placing windows around regions in the image that are classified as worthy of attention. The selection of which regions to so classify depends on the goal and priorities of the current level 4 task, or on the detection of noteworthy attributes of groups of level 3 regions. The shape of the windows is determined by the set of pixels in the recognized object entity image. The size of the windows is determined by the confidence factor generated by level 4 recursive estimation operations.

Level 4 Grouping

At level 4, recognized surface entities in the same class are grouped into level 4 regions, or object entities, based on gestalt properties such as contiguity, similarity, proximity, pattern continuity, and symmetry. Surfaces in the same class that are contiguous along their boundaries

and have similar range and velocity may be grouped into object entities. Boundaries that separate surfaces with different range, velocity, average surface orientation, and average color are used to distinguish between different objects. Level 4 grouping is a hypothesis that all the pixels in the region have the common property of imaging the same physical object in the real world. This grouping hypothesis is confirmed or rejected by the level 4 filtering function.

Level 4 Computation

For each of the hypothesized object entities, entity attributes such as position, range, and motion of the center-of-gravity, average surface texture, average color, solid-model shape, projected area, and estimated volume can be computed. For each hypothesized object entity, computed attributes fill slots in an observed object entity frame.

Level 4 Filtering

A recursive estimation process compares observed object entity attributes with predictions based on estimated object entity attributes. Included in the prediction process are the estimated motions of object entities, and expected results of commanded actions. Comparison of observed entity attributes with predicted entity attributes produces correlation and difference values. Difference values are used to update the attribute values in the estimated object entity frames. A confidence level is computed as a function of the correlation and difference values. If the confidence level of the recursive estimation filter rises above threshold, the object entity grouping hypothesis is confirmed.

Level 4 Classification

The classification process compares the attributes of each confirmed object entity with attributes of object entity class prototypes stored in the knowledge database. A match causes a confirmed geometrical object to be classified as a generic, or a specific object. For a ground vehicle performing a typical task, a list of generic object classes may include the ground, the sky, the horizon, dirt, grass, sand, water, bush, tree, rock, road, mud, brush, woods, log, ditch, hole, pole, fence, building, truck, tank, etc. There may be several tens, hundreds, or even thousands of generic object classes in the KD. A list of specific object classes may include a specific tree, bush, building, vehicle, road, or bridge. As a result of classification, a match between an object entity frame and a labeled object entity image is formed in which each pixel has a pointer to the object entity class to which it belongs.

Summary of Level 4:

(1) Portions of the image containing object entities of attention are windowed.
(2) Surface entities with similar attributes are tentatively grouped into level 4 regions, or object entities.
(3) The attribute values of each hypothesized object entity are computed.
(4) Attributes of each hypothesized object entity are estimated by recursive estimation and each grouping hypotheses is either confirmed or rejected.

(5) The attributes of each confirmed object entity are compared with the attributes of a set of object entity class prototypes, and those that match are assigned to object entity classes. Object entity frames are given pointers to the class to which they belong.

Level 5 Regions: Section Entities and Events

At level 5 of the SP hierarchy, groups of object entities with similar range, motion, and other attributes are grouped, processed, and analyzed as a group, collection, or assemblage of objects. For the Demo III Scout Platoon scenario, the smallest group is called a Section. Attributes of object entities comprising each section entity are combined into section entity attributes. Also at level 5, temporal strings of level 4 events are integrated into level 5 events.

Level 5: Windowing

At level 5, windowing consists of placing a window around regions in the image that are classified as section entities-of-attention. The selection of which regions to so classify depends on the task, and on detection of entity attributes that are worthy of attention. The shape of the windows is determined by the collection of pixels in a confirmed section entity image. The size of the windows is determined by the confidence factor associated with the level 5 recursive estimation process.

Level 5: Grouping

Object entities in the same class are grouped into level 5 regions, or section entities, based on gestalt properties such as contiguity, similarity, proximity, pattern continuity, and symmetry. Objects in the same class that are near each other and have similar range and velocity may be grouped into section entities. These grouping hypotheses will be confirmed or rejected by the level 5 filtering function.

Level 5: Computation

For each of the hypothesized section entities, attributes such as position, range, and motion of the center-of-gravity, density, and shape can be computed. These computed attributes become observed entity attributes in an observed section entity attribute frame.

Level 5: Filtering

A recursive estimation process compares observed section entity attributes with predictions based on estimated section entity attributes and expected results of commanded actions. Comparison of observed section entity attributes with predicted entity attributes produces correlation and difference values. Difference values are used to update the attribute values in the estimated section entity frames. A confidence level is computed as a function of the correlation and difference values. If the confidence level of the recursive estimation filter rises above threshold, the section entity grouping hypothesis is confirmed.

Level 5: Classification

The classification process compares the attributes of each confirmed section entity with attributes of section entity class prototypes stored in the knowledge database. A match causes a geometric section entity to be classified as a generic, or a specific section entity in the world model knowledge database. For example in the case of a ground vehicle, a list of generic section classes may include a woods, fields, groups of vehicles, groups of people, and clusters of buildings. A list of specific section classes may include a specific group of humans, trees, bushes, buildings, vehicles, or the intersection of two or more roads.

As a result of classification, a new class image is formed in which each pixel has a pointer to (or the name of) the generic or specific section entity class to which it belongs. There may be several tens, hundreds, or even thousands of section classes in the KD.

Summary of Level 5 SP:

(1) Portions of the image containing section entities of attention are windowed.
(2) Object entities with similar attributes are tentatively grouped into level 5 regions, or section entities.
(3) The attribute values of each observed section entity are computed.
(4) Attributes of each hypothesized section entity are estimated by recursive estimation and each grouping hypotheses is either confirmed or rejected.
(5) The attributes of each confirmed section entity are compared with the attributes of a set of section entity class prototypes, and those that match are assigned to section entity classes. Object entity frames are given pointers to the class to which they belong.

5.0 GENERIC SHELL

It is clear from the foregoing discussion that there are many differences between nodes at different levels of the 4D/RCS architecture as well as between different units within the organization. There are differences in the nature of the data, the range and resolution of space and time, the scope of planning, and the level of detail in plans and actions. There are differences in responsibilities, and in the kinds of knowledge, skills, and abilities. But, there are also many features of 4D/RCS nodes that are common between nodes at all levels and in all organizational units. For example, all nodes function as augmented state-machines. When triggered, they read their inputs, compute a set of functions, make decisions, write outputs, and wait for the next trigger. When triggered, all nodes run to completion and wait until the next trigger. All nodes can have built-in diagnostics that describe their current state, their current inputs, outputs, and a trace of recent history. All nodes can keep statistics regarding execution time and other parameters.

This commonality suggests that a Generic Shell might be constructed that could serve as a software development tool as well as serve as a basis for debugging and diagnostic procedures. Figure 43 shows an example of a Generic Shell for Behavior Generation at the Servo level for the Velocity Subsystem. Commands to the Servo level indicate the desired acceleration and yaw rate to be achieved 50 ms in the future. The Job Assignor decomposes this task into a commanded position for the steering wheel actuator, a commanded brake pedal force, a commanded throttle position, and a commanded gear shift setting – all to be achieved 50 milliseconds in the future. For each actuator, a Scheduler generates a schedule of planned actions and planned states over the 50 ms interval. Selected job plans are placed in a plan data structure in the Executor where they are accessed by a plan sequencer. On each cycle of the controller, each executor selects a planned action and a desired resulting state from its respective plan (stored as a state table). The planned action is sent to the control law as a feed forward action. Simultaneously, the desired resulting state is compared with the feedback predicted state from the world model. The difference between the feedback predicted state and the planned state is a feedback error that is also sent to the control law where compensation is computed and added to the feed forward planned action.

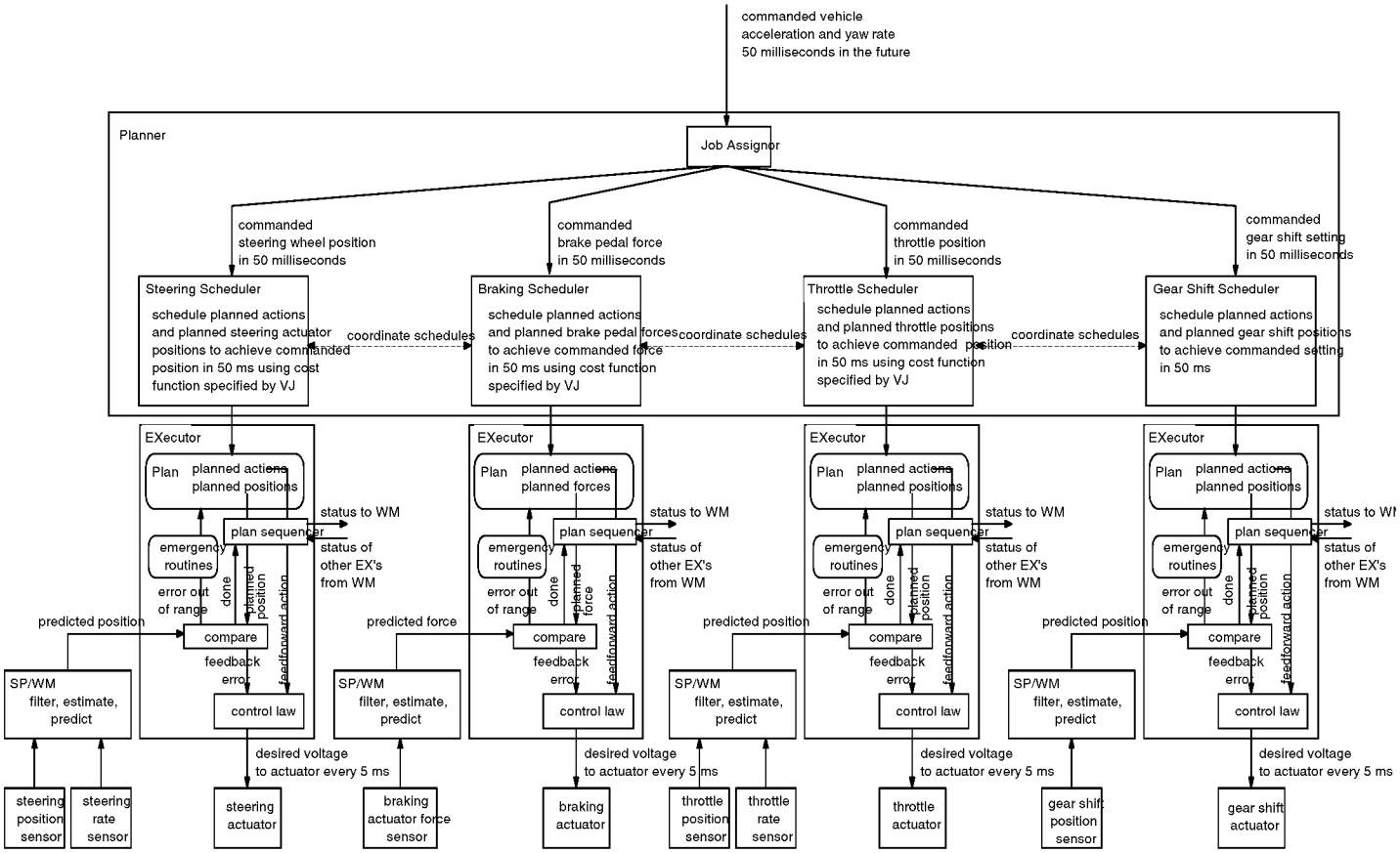


Figure 43. An example of the 4D/RCS Generic Shell for behavior generation at the Servo level.

A generic shell for behavior generation makes it relatively easy to program at this level of complexity. The generic shell provides the timing and communication functions and allows the system designer to think in terms of planning and control algorithms rather than C++ code constructs. A number of software engineering tools have been developed for constructing generic shell modules. These range in degree of formality from C++ templates to Unified Modeling Language (UML) and Architectural Description Languages (ADL) [Huang 2001] [Messina 2000] [Dabrowski 1999]. A tool developed by John Horst at NIST uses Control Shell [Horst 00]. Another tool developed by Will Shackleford at NIST is written in Java. [Gazi01] Additional tools are being developed at Ohio State University using a LabView front end, [Gazi01] and by Pathway Technologies using MatLab as a front end [Anathakrishnan02][9]

Figure 44 shows the organizational structure for behavior generation generic shells at the Primitive (Prim) and Subsystem levels integrated with those at the Servo level. The Driving Subsystem at the Subsystem level is decomposed into the Camera Subsystem and the Vehicle Subsystem at the Prim level, and into the Pan/Tilt Subsystem and the Velocity Subsystem at the

[9] [9] Commercial equipment and materials are identified in order to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

Servo level. The Servo planner looks ahead 50 ms in the plan generated by the Prim level. The Prim planner looks ahead 500 ms in the plan generated by the Subsystem level. The Subsystem level planner looks ahead 5 seconds in the plan generated by the Vehicle level. At each level, the planner always looks ahead at least to its own planning horizon in the higher level plan.

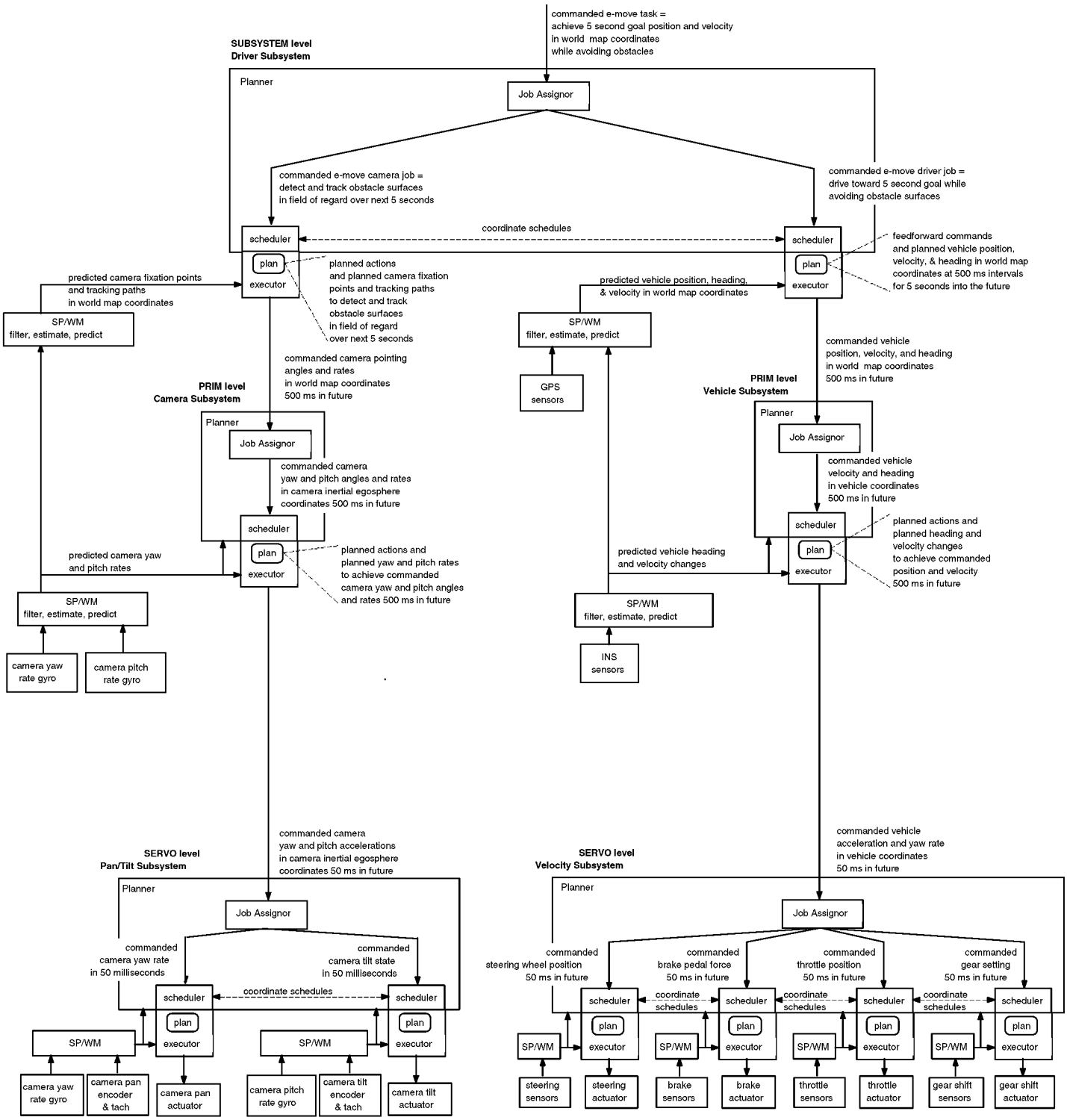


Figure 44. Generic shell BG modules at the Subsystem, Primitive, and Servo levels.

6.0 SUMMARY AND CONCLUSION

The 4D/RCS reference model architecture will evolve as technology and standards advance. Over the past five years, it has been implemented on the Demo III experimental unmanned ground vehicles with great success. However, the implementation lags behind architecture specification. Not all of the features described in this document have been implemented. The architecture is most fully implemented in the Behavior Generation hierarchy. Multi-level, multi-resolution, real-time planning has been implemented very successfully at the vehicle level and below. The world model consists mainly of terrain elevation maps with labeled pixels. Map pixels have been classified as road, grass, water, and obstacles with attributes of slope and roughness. A priori maps contain information about roads, streams, buildings, and wooded areas. As yet, little has been done in grouping pixels into entities or signals into events, and no significant relationships have been established that describe situations. Work has begun on the representation of moving objects. Very little has been implemented that corresponds to the section level or above. It is expected that as work continues, some features may be added to the architecture, while others may be modified or deleted. However, the work on Demo III to date suggests that the 4D/RCS architecture is fundamentally sound and a reliable roadmap for system development and integration.

The 4D/RCS reference model architecture is naturally adaptable to the DoD/Army standards in a combined domain of vehicle systems, combat support, and software engineering. 4D/RCS provides an architectural framework to facilitate component and interface standards development, including command and control, sensors, communication, mapping, operating environments, safety, security, software engineering, user interface, data interchange, and graphics. As such, the 4D/RCS reference model architecture forms an architectural framework for standards.

PRODUCT/COMPANY DISCLAIMER

Commercial equipment and materials are identified in order to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the equipment or materials identified are necessarily the best available for the purpose.

7.0 REFERENCES

E.H. Adelson and J.A. Movshon, (1982) "Phenomenal coherence of moving visual patterns." *Nature, 30*, 523-525.

J.S. Albus, (1991) "Outline for a Theory of Intelligence," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 21, No. 3, pgs. 473-509, May/June.

J.S. Albus, (1975a) "A New Approach to Manipulator Control: The Cerebellar Model Articulation Controller (CMAC)," *Journal of Dynamic Systems, Measurement and Control*, American Soc. of Mechanical Engineers, Sept.

J.S. Albus, (1975b) "Data Storage in the Cerebellar Model Articulation Controller (CMAC)" *Journal of Dynamic Systems, Measurement and Control*, American Soc. of Mechanical Engineers, Sept.

J. S. Albus and T. H. Hong, (1990) "Motion, Depth, and Image Flow," *Proceedings of IEEE 1990 Int'l Conference on Robotic and Automation*, May.

J.S. Albus and A.M. Meystel, (2001) *Engineering of Mind: An Introduction to the Science of Intelligent Systems*, John Wiley and Sons.

J. S. Albus and A. M. Meystel, (1996) "A Reference Model Architecture for Design and Implementation of Intelligent Control in Large and Complex Systems," *International Journal of Intelligent Control and Systems*, Vol.1, No. 1, pp15-30.

J. Albus, K. Murphy, A. Lacaze, S. Legowik, S. Balakirsky, T. Hong, M. Shneier, E. Messina, (2002) "4D/RCS Sensory Processing and World Modeling on the Demo III Experimental Unmanned Ground Vehicles." *17th IEEE International Symposium on Intelligent Control*, Vancouver, British Columbia, Canada, October 27-30.

S. Anathakrishnan, S. Agrawal, R. Venugopal, M. Demeri (2002) "RCS Based Hardware-in-the-Loop Intelligent System Design and Performance Measurement," *Proceedings of the Performance Measures for Intelligent Systems (PerMIS) Workshop*, National Institute of Standards and Technology, Gaithersburg, MD, August 13-15

Army (2000), Army Science Board FY2000 Summer Study Final Report on "Technical and Tactical Opportunities for Revolutionary Advances in Rapidly Deployable Joint Ground Forces in the 2015-2025 Era" Volume II, Operations Panel Report, pg.35.

ATR (2002) http://atrcorp.com/~

S. Balakirsky, O. Herzog, (2002) "Planning with Incrementally Created Graphs," NISTIR, Gaithersburg, MD.

S. Balakirsky, E. Messina, J. Albus, (2002) "Architecting a Simulation and Development Environment for Multi-Robot Teams." *Proceedings of the International Workshop on Multi Robot Systems*, Washington, DC, March 18-20.

S. Baten, R. Mandelbaum, M. Lutzeler, P. Burt, and E. D. Dickmanns, (1998) "Area-based stereo image processing within the 4D architecture for autonomous off-road navigation," AutoNav Meeting, FL, January, 1998.

Boeing (2002) http://www.boeing.com/defense-space/ic/fcs/bia/flash.html

J. A. Bornstein, (2002) "U.S. Army ground robotics research program," *Proceedings of SPIE Aerosense Conference Vol. 4715,* Orlando, Florida, 1-5 April.

Brooks, R.A. 1991. Intelligence Without Representation. Artificial Intelligence 47: 139-159

Brooks, R.A. 1986. A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation RA-2(1): 14-23.

P. J. Burt, (1989) "Multiresolution Techniques for Image Representation, Analysis, and 'Smart' Transmission," *SPIE Conf. 1199: Visual Communications and Image Processing IV*, Philadelphia, Nov.

P. J. Burt, (1988) "Attention Mechanisms for Vision in a Dynamic World," *IEEE.*

P. J. Burt and E. H. Adelson (1983) "The Laplacian Pyramid as a Compact Image Code," *IEEE Transactions on Communications*, VOL. COM-31, No. 4, April.

P.J. Burt and G. van der Wal, (1990) "An Architecture for Multiresolutional Focal, Image Analysis," *Proceedings of 10th International Conference on Pattern Recognition*, Atlantic City, June 16-21.

P. J. Burt, et al., (1989) "Object Tracking with a Moving Camera: An Application of Dynamic Motion Analysis," *Proceedings of the Workshop on Visual Motion*, Irvine, CA, March 20-22, 1989.

CISA (2002) http://www.fas.org/irp/program/core/c4isr.htm

DoD (2002) http://www-jta.itsi.disa.mil/

Dabrowski, C., Huang, H.M., Messina, E., Horst, J.A., (1999) Formalizing the NIST 4-D/RCS Reference Model Architecture Using an Architectural Description Language, NISTIR 6443, National Institute of Standards and Technology, Gaithersburg, MD.

E. D. Dickmanns, (1999) "An Expectation-based, Multi-focal, Saccadic (EMS) Vision System for Vehicle Guidance," *Proceedings 9-th Int. Symp. on Robotics Research (ISRR'99)* Salt Lake City, October.

Dickmanns, E. (1996) From a presentation given at NIST in December.

E. D. Dickmanns, (1995) "Parallel Use of Differential and Integral Representations for Realizing Efficient Mobile Robots," *7th ISRR*, Munich, Germany.

E. D. Dickmanns, et al.., (1994) "The Seeing Passenger Car 'VaMoRs-P'," *International Symposium on Intelligent Vehicles'94*, Paris, October 24-26.

E. D. Dickmanns, (1992a) "Machine Perception Exploiting High-Level Spatio-Temporal Models," *AGARD Lecture Series 185: Machine Perception*; Hampton, VA; Munich; Madrid, Sept.

E. D. Dickmanns, (1992b) "A General Dynamic Vision Architecture for UGV and UAV," *Journal of Applied Intelligence* 2.251-270 (1992), 1992 Kluwer Academic Publishers, Boston, Manufactured in The Netherlands.

E.D. Dickmanns and V. Graefe (1988) a) "Dynamic monocular machine vision," b) "Application of dynamic monocular machine vision." *Journal of Machine Vision & Applications*, Springer-Int, Nov. 1988, pp 223-261.

J. M. Evans, E. R. Messina, J. S. Albus, C. I. Schlenoff, (2002) "Knowledge Engineering for Real Time Intelligent Control." *Kowledge-Based Intelligent Information and Engineering Systems Conference*, Italy, September.

L. Fogel (1999) Intelligence Through Simulated Evolution, Wiley, NY.

Gazi, V., Moore, M.L., Passino, K.M., Shackleford, W.P., Proctor, F.M., and Albus, J.S., (2001) The RCS Handbook, Wiley Series on Intelligent Systems,John Wiley and Sons, Inc., New York, New York.

J. Gibson (1950) *The Perception of the Visual World,* Houghton-Mifflin, Boston, MAJ. Gibson, P. Olum, and F. Rosenblatt (1955) "Parallax and perspective during aircraft landings." *American Journal of Psychology, 68,* pp. 327-385.

J. A. Haas, P. David, and B. T. Haug, "Target Acquisition and Engagement from the Unmanned Ground Vehicle: the Robotic Test Bed of Demo II," ARL-TR-1063, Army Research Lab., March 1996.

T. Hong, S. Balakirsky, E. Messina, T. Chang, M. Shneier, (2002a) "A Hierarchical World Model for an Autonomous Scout Vehicle." *SPIE Aerosense Symposium*, Orlando, FL, April.

T. Hong, T. Chang, C. Rasmussen, M. Shneier, (2002b) "Feature Detection and Tracking for Mobile Robots Using a Combination of Ladar and Color Images." *IEEE International Conference on Robotics and Automation*, Washington DC.

T. Hong, C. Rasmussen, T. Chang, and M. Shneier, (2002c) "Fusing Ladar and Color Image Information for Mobile Robot Feature Detection and Tracking." *Proc. 7th Inter. Conf. On Intelligent Autonomous Systems*, Marina del Rey, CA, March.

T. Hong, C. Rasmussen, T. Chang, and M. Shneier, (2002d) "Road Detection and Tracking for Autonomous Mobile Robots." *SPIE Aerosense Symposium*, Orlando, FL, April.

Horst, J., (2000) "Architecture, Design Methodology, and Component-Based Tools for a Real-Time Inspection System", The 3rd IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC 2000), Newport Beach, CA, March 15-17.

Hui-Min Huang, Elena Messina, Harry Scott, James Albus, Frederick Proctor, and William Shackleford, (2001) "Open System Architecture for Real-time Control Using a UML Based Approach," First Workshop on Describing Software Architecture with UML, Held in conjunction with the 23rd ISCE, Toronto, Canada.

Hui-Min Huang, Harry Scott, Elena Messina, Maris Juberts, and Richard Quintero, (2000) "Intelligent System Control: A Unified Approach and Applications," Book Chapter in Academic Press Volumes on "Expert Systems Techniques and Applications," C. T. Leondes, Ed.

Huang, H. and Messina, E., (1996) "NIST-RCS and Object-Oriented Methodologies of Software Engineering: A Conceptual Comparison," Proceedings of the Intelligent Systems: A Semiotic Perspective Conference, NIST, Gaithersburg, October.

JAUGS (2002) http://www.jaugs.org/

JTA-A (2002) http://akea-cio.army.mil/jtaa/documents.asp

Kalman, R.E., (2002) http://www.cs.unc.edu/~welch/kalman/

Koenderink, J. (1990) *Solid Shape*, MIT Press, Cambridge, MA

Koestler, A., (1967) *The Ghost in the Machine*, Random House, NY.

R. Mandelbaum and P. Anandan, (1998) "AutoNav UBM-Sarnoff Collaboration: Interface Control Document," January.

Y. Maximov, and A. Meystel, (1992) Optimum Design of Multiresolutional Hierarchical Control Systems, Proceedings of IEEE International symposium on intelligent control, pg 514-520, Glasgow, UK.

A.M. Meystel and J.S. Albus, (2002) *Intelligent Systems: Architecture, Design, and Control*, John Wiley and Sons.

Messina, E. and Huang, H. (2002), "Specifications for Intelligent Systems: How do they differ from those of non-intelligent ones?" Proceedings of Knowledge Engineering Workshop,

Messina, E., Huang, H.M., Scott, S.A., (2000) An Open Architecture Inspection System, Proceedings of the Japan-USA Symposium on Flexible Automation, Ann Arbor, MI.

Messina, E., Horst, J.A., Kramer, Tom , Huang, H.M., Michaloski, J.L., (1999) Component Specifications for Robotics Integration, Autonomous Robots, May 1999 Issue.

K. Murply, M. Abrams, S. Balakirsky, T. Chang, T. Hong, A. Lacaze, and S. Legowik, (2002) "Intelligent Control For Off-Road Driving." *Presented at First International NAISO Controess on Autonomous Intelligent Systems*, Deakin University, Geelong, Australia, February.

K. Murphy and S. Legowik, (1996) "GPS Aided Retrotraverse For Unmanned Ground Vehicles," Proceedings of the SPIE 10th Annual AeroSense Symposium, Conference 2738, Navigation and Control, Technologies for Unmanned Systems, Orlando, FL.

OTB (2002) http://www.onesaf.org/publicotb1.html

PEO C3T (2002) http://www.monmouth.army.mil/newpages/vCpeoc3s.html

C. Rasmussen, (2001) "Laser Range-, Color-, and Texture-based Classifiers for Segmenting Marginal Roads." *Proc. Conf. On Computer Vision & Pattern Recognition Technical Sketches*, Kauai, HI, December.

C. Rasmussen, (2002) "Combining Laser Range, Color, and Texture Cues for Autonomous Road Following." *Proc. IEEE Inter. Conf. On Robotics & Automation*, Washington, DC, May

C. Schlenoff, (2002) "Linking Sensed Images to an Ontology of Obstacles to Aid in Autonomous Driving." *Proceedings of the AAAI-2002 Workshop on Ontologies and the Semantic Web*, Edmonton, Canada, July 28 - August 1.

Schlenoff, C., Gruninger M., Tissot, F., Valois, J., Lubell, J., Lee, J., (2000) The Process Specification Language (PSL): Overview and Version 1.0 Specification, NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD

C. Shoemaker, J. Bornstein, S. Myers, and B. Brendle, (1999) "Demo III: Department of Defense testbed for unmanned ground mobility", *Proceedings of SPIE Vol. 3693, AeroSense Session on Unmanned Ground Vehicle Technology*, Orlando, Florida, April 7-8.

Shoemaker, C. M. and Bornstein, J. A. 1998. Overview of the Demo III UGV Program. Proc. Of the SPIE Robotic and Semi-Robotic Ground Vehicle Technology, Vol. 3366, pp. 202-211.

R. Sutton and A. Barto (1998) Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning), MIT Press, Cambridge, MA.

TRM (2002) http://www-trm.itsi.disa.mil.

C. D. Tomlin (1990) *Geographic Information Systems and Cartographic Modeling,* Prentice Hall, Englewood Cliffs, NJ.

VRA (2001) Vehicle Electronics (VETRONICS) Reference Architecture (VRA), Working Draft 0.8, TARDEC, Warren, MI.

Vetronics (2002) http://www.tacom.army.mil/tardec/vetronics/

M. Wertheimer, (1958) Principles of perceptual organization, in "Reading in perception," D.C. Beardslee and M. Wertheimer (eds.), Van Nostrand-Reinhold, Princeton, New Jersey.

WSTAWG (2002) http://wstawg.army.mil/