RCS: An Intelligent Agent Architecture

Jim Albus, Tony Barbera, Craig Schlenoff

National Institute of Standards and Technology (NIST)
100 Bureau Drive, Stop 8230
Gaithersburg, MD 20899
(james.albus, tony.barbera, craig.schlenoff) @nist.gov

Abstract

RCS (Real-time Control System) is an intelligent agent architecture designed to enable any level of intelligent behavior, up to and including human levels of performance. RCS was inspired 30 years ago by a theoretical model of the cerebellum, the portion of the brain responsible for fine motor coordination and control of conscious motions. It was originally designed for sensory-interactive goal-directed control of laboratory manipulators. Over three decades, it has evolved into a real-time control architecture for intelligent machine tools, factory automation systems, and intelligent autonomous vehicles.

In this paper, we describe the 4D/RCS architecture, how it relates to other popular intelligent agent architectures, how it addresses the three most significant theoretical arguments against intelligent agent architectures, and its underlying engineering methodology.

1.0 Introduction

Interest in intelligent agent architectures has grown rapidly over the past two decades as a result of a confluence of three important events:

1. **The emergence of a computational theory of intelligence:** A fully developed scientific theory of intelligence does not yet exist, but an understanding of how to build intelligent systems is developing faster than most people appreciate. Progress is rapid in many different fields. Recent results from a number of different disciplines, including the neurosciences, cognitive psychology, artificial intelligence, robotics, and intelligent machines, have laid the foundations for a computational theory of intelligence [3].
2. **The continued exponential growth of computing power:** The estimated computational power of the human brain is already rivaled by existing supercomputers. Within the next quarter century, computational power approaching that of the human brain can be expected from a small network of desktop machines [17]. This means that serious attempts can be made to model the functional capabilities of the brain in perception, cognition, and behavioral skills. Of course, having the computational power is only part of the challenge; the rest is making proper use of it.
3. **Growth in user interest for military and commercial applications:** Potential applications in both civilian and military systems have begun to emerge for the control of autonomous vehicles. In the United States, military interest in unmanned vehicles (air, ground, and sea) has grown rapidly as autonomous vehicle capabilities have been demonstrated that far exceed previous expectations [24]. In Japan, Europe, and the U.S., automotive companies are actively pursuing commercial applications of adaptive cruise control, crash warning systems, and collision avoidance technology. The eventual result may be the intelligent autonomous automobile.

However, the increased interest in intelligent agent architectures has resulted in various disparate approaches to their development and implementation. In this paper, we discuss one of those approaches, the 4D/RCS (Real-time Control System) and compare it to some of the other architectures being developed. We also show how 4D/RCS addresses some of the major theoretical arguments against intelligent agent architectures, namely, abductive inference, symbol grounding, and the frame problem.

Section 2 will discuss the characteristics of various intelligent agent architectures, Section 3 describes the 4D/RCS reference model, Section 4 describes how 4D/RCS addresses some of the theoretical problems of computational models, Section 5 steps through the RCS methodology for system design, and Section 6 concludes the paper.

2.0 Intelligent agent architectures

An intelligent agent architecture can be defined as the organizational structure of functional processes and knowledge representations that enable the modeling of cognitive phenomena. Over the past half-century, several intelligent agent architectures have been developed. One of the earliest was the ACT architecture [8]. ACT grew out of research on human memory. Over the years, ACT has

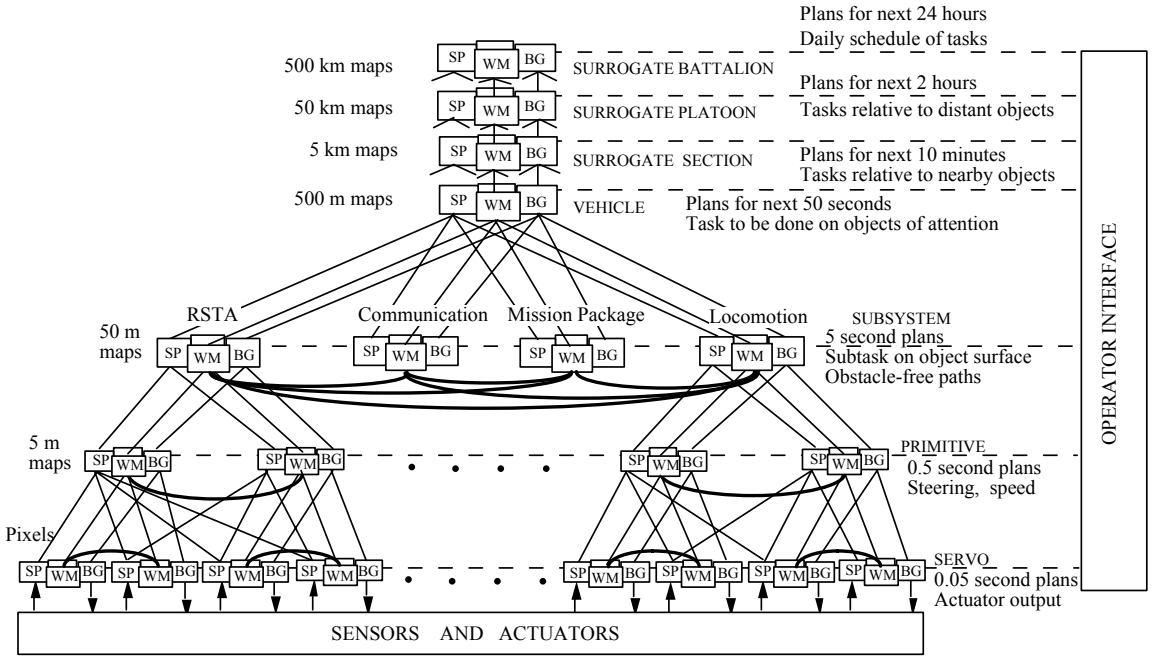

Figure 1. A 4D/RCS reference model architecture for an autonomous vehicle.

evolved into ACT* and more recently, ACT-R. ACT-R is being used in several research projects in an Advanced Decision Architectures Collaborative Technology Alliance for the U.S. Army [15]. ACT-R is also being used by thousands of schools across the country as an algebra tutor – an instructional system that supports learning-by-doing.

Another well-known and widely used architecture is Soar. Soar grew out of research on human problem solving, and has been used for many academic and military research projects in problem solving, language understanding, computational linguistics, theorem proving, and cognitive modeling [18]. A recent commercial version of Soar (Tac-Air Soar) has been developed to address a number of simulation and training problems for the U.S. Air Force[1] [1].

Other intelligent agent architectures include Prodigy, ICARUS, IMPRINT, EPIC, and RCS. Like Soar, Prodigy uses search through a problem space to achieve goals cast as first-order expressions [19]. ICARUS is a reactive architecture that encodes knowledge as reactive skills [23]. IMPRINT is a task description language designed for the Army to capture the procedural specification of tactical behavior scenarios. It contains a dynamic, stochastic, discrete-event network modeling tool designed to help assess the interaction of soldier and system performance throughout the system lifecycle – from concept and design through field testing and system upgrades. IMPRINT has been integrated with ACT-R to model military behaviors [9]. EPIC is an architecture that models the detailed timing of human perceptual, cognitive, and motor activity, including the input/output characteristics of the nervous system connecting the higher level cognitive functions to the external world [16]. RCS is a control system architecture inspired by a theory of cerebellar function [2]. RCS models the brain as a hierarchy of goal-directed sensory-interactive intelligent control processes that theoretically could be implemented by neural nets, finite state automata, or production rules [7].

RCS is similar to other intelligent agent architectures in that it represents procedural knowledge in terms of production rules, and represents declarative knowledge in abstract data structures such as frames, classes, and semantic nets. RCS differs from other intelligent agent architectures in that it also includes signals, images, and maps in its knowledge database, and maintains a tight real-time coupling between iconic and symbolic data structures in its world model. RCS is also different in: a) its level of specificity in the assignment of duties and responsibilities to agents and units in the behavior generating hierarchy; b) its emphasis on controlling real machines in real-world environments; and c) its malleability to the specific requirements of the domain. To elaborate on (c), when using other architectures, one is often getting a monolithic piece of software and then adds knowledge, often in the form of rules and chunks. Within 4D/RCS, one is able to apply appropriate planning systems, sensory processing

[1] The name of commercial products or vendors does not imply NIST endorsement or that this product is necessarily the best for the purpose.

algorithms, and capture knowledge in a format that is most conducive to the domain of interest. RCS does not dictate what form that knowledge must take. Instead, it provides a framework that allows one to apply whatever knowledge representation technique that is most appropriate for the knowledge that is to be represented.

RCS evolved from the bottom up as a real-time intelligent control system for real machines operating on real objects in the real world. The first version of RCS was developed as a sensory-interactive goal-directed controller for a laboratory robot. Over the years, RCS has evolved into an intelligent controller for industrial robots, machine tools, intelligent manufacturing systems, automated general mail facilities, automated stamp distribution systems, automated mining equipment, unmanned underwater vehicles, and unmanned ground vehicles [4]. Throughout its development, all symbols in the RCS world model have been grounded to objects and states in the real world.

The most recent version of RCS (4D/RCS) embeds elements of Dickmanns [13] 4-D approach to machine vision within the RCS control architecture. 4D/RCS was designed for the U.S. Army Research Lab AUTONAV and Demo III Experimental Unmanned Vehicle programs and has been adopted by the Army Future Combat System program for Autonomous Navigation Systems [5].

3.0 The 4D/RCS Reference Model Architecture

A reference model architecture describes the functions, entities, events, relationships, and information flow that takes place within and between functional modules. A reference model provides a framework for the specification of functional requirements, the design of software to meet those requirements, and the testing of components and systems. A block diagram of a 4D/RCS reference model architecture is shown in Figure 1. Each node in the architecture represents an operational unit in an organizational hierarchy. Each node contains a behavior generation (BG), world modelling (WM), sensory processing (SP), and value judgment (VJ) processes together with a knowledge database (KD) (not shown in Figure 1.)

Each node contains both a deliberative and a reactive component. Bottom-up, each node closes a reactive control loop driven by feedback from sensors. Top-down, each node generates and executes plans designed to satisfy task goals, priorities, and constraints conveyed by commands from above. Within each node, deliberative plans are merged with reactive behaviors.

Each 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). Task commands from a supervisor BG process are sent to a planner module that decomposes each task into a set of coordinated plans for subordinate BG processes. For each subordinate there is an Executor that issues commands, monitors progress, and compensates for errors between desired plans and observed results. The Executors use feedback to react quickly to emergency conditions with reflexive actions. Predictive capabilities provided by the WM may enable the Executors to generate preemptive behavior.

SP and WM processes interact to support windowing (i.e., attention), grouping (i.e., segmentation), recursive estimation (e.g., Kalman filtering), and classification (i.e., detection or recognition). WM processes generate and update images, maps, entities, events, attributes, and states in the KD. Working together, BG, WM, and SP 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.

At the top, the highest-level task is defined by the highest-level (i.e., mission) goal. At each successive level in the hierarchy, commanded tasks from above are decomposed into subtasks that are sent to subordinates below. Finally, at the bottom, subcommand outputs are sent to actuators to generate forces and movements. Also at the bottom, sensors transform energy into signals that provide sensory input.

4.0 Discussion

4D/RCS addresses three of the most significant theoretical arguments raised against the possibility of computers achieving human levels of intelligence. These are:

4.1. Abductive Inference

Abductive inference is the process of reasoning backward from consequent to antecedent. It has been described by Pierce [20] as “nothing but guessing.” The inability of local syntactical systems to perform abductive inference is cited by Fodor [14] as why he believes computational processes cannot produce true intelligence. To Fodor, all computer operations are inherently local and syntactic, and hence fundamentally incapable of context sensitive logic.

But the RCS architecture is driven top-down by high level mission goals, priorities, and constraints. These provide global context for making gestalt hypotheses (i.e., perceptual guesses) regarding where to focus attention and how to group (or segment) signals and pixels into patterns and regions that correspond to entities and events in the external world. At each level of sensory processing, abductive inferences in the form of gestalt hypotheses are used to segment signals and images. Abductive inferences can be tested by comparing expectations based on hypotheses against observations from sensors. For each

hypothesized entity or event, variance between predictions and observations provides a measure of confidence in the hypothesis. When variance is small, confidence is high, and vice versa. If confidence falls below threshold, a hypothesis is rejected and another generated. This supports Pierces's claims that abduction can be represented in a "perfect definite logical form."

4.2 Symbol Grounding

Symbol grounding is the establishment of direct correspondence between internal symbolic data and external real world entities, events, and relationships. The inability of local syntactical systems to perform symbol grounding is cited by Searle [22] as why he believes computational processes can never be really intelligent. To Searle, computer operations are without semantic meaning because the symbols they manipulate are never grounded in the real world.

But the 4D/RCS architecture establishes and maintains a direct link between the internal world model and the external real world. An RCS attention process directs sensors toward regions of the world that are important. An RCS segmentation process applies context-sensitive gestalt grouping hypotheses to patterns of signals from sensors. As a result of segmentation, spatial and temporal groupings are linked to named symbolic data structures (such as C structs, or C++ objects and classes) that represent hypothesized entities and events. Geometric and temporal attributes of hypothesized groups are computed, and relationships (represented as pointers) between entities, events, and their constituent elements are established and maintained. Finally, entities and events are classified and recognized by comparing observed attributes to stored attributes of class prototypes. This entire process is repeated at each stage of sensory processing at a rate fast enough to capture the dynamics of the entities and events being attended to.

This recursive two-way interaction between model-based expectations and sensory-based observations provides symbol grounding. Expectations based on attributes and class membership of entities and events in the world model are constantly compared against observations derived from sensors monitoring corresponding entities and events in the real world. In this way, symbolic representations of entities, events, and relationships in the 4D/RCS world model are grounded to entities, events, and relationships in the real world.

4.3 The Frame Problem

The frame problem is the problem of predicting what in the world changes as the result of an action, and what stays the same. The frame problem results from attempting to model the world entirely in terms of logical propositions [21]. Many important features about the world (in particular geometry and dynamics) are not easily modeled by logical propositions. For example, as I write this, I am pondering the difficulties of using logical propositions to model a bookcase in my office that is filled with books, papers, boxes, folders, and assorted junk and trash. As I ponder this, a fly maneuvers at high speed (in terms of body lengths per second) around my bookcase without danger of collision. Apparently, the fly's internal model of the world enables it to fly between shelves and stacks of books and papers without collision, and to land on the tip of a straw in an old soda can without difficulty. Surely the fly's brain does not represent my bookcase in terms of logical propositions.

It has been said that a picture is worth a thousand words. I would venture that a 4-D representation (3 spatial + 1 temporal) of a complex scenario in a dynamic environment may be worth a million logical propositions.

On the other hand, the location and direction of motion of objects in the world are easily represented in an image or map, and distinguishing what changes from what does not in a dynamic environment can be easily determined by simple comparison between one visual image and the next. That is why 4D/RCS representations include iconic or metrical formats such as visual images and maps in addition to symbolic data structures such as frames, objects, classes, and rules. Both iconic and symbolic formats are linked together by pointers in a real-time relational database that is updated in each node at a rate that is commensurate with the requirements of the planning and control processes within that node.

At the lowest level, signals and images from sensors are sampled many times per second and compared with expectations generated from world model predictions. At all levels, differences between observations and expectations are used to update the internal model by a process of recursive estimation. Predictions from the world model are projected back into sensor coordinates, overlaid on, and cross correlated with images and maps of the external world. The result is that symbols in the world model are linked to, and can be projected back onto, pixels and regions in images and maps. In this way, the internal world model is effectively servoed to the external real world, and symbolic data structures in the world model are grounded to entities and events in the real world. Thus, the ability for 4D/RCS to accommodate and integrate multiple types of representations into a common architecture takes major steps in addressing the frame problem [6].

At each echelon of the behavior generation hierarchy, the range and resolution of knowledge about the world is defined by the requirements of the task. At each echelon, an attention process, driven by task priorities and differences between observations and predictions, selects those regions of the world that are important to the task. At each echelon, the world model enables 4D/RCS behavior generation processes to plan tasks and paths that

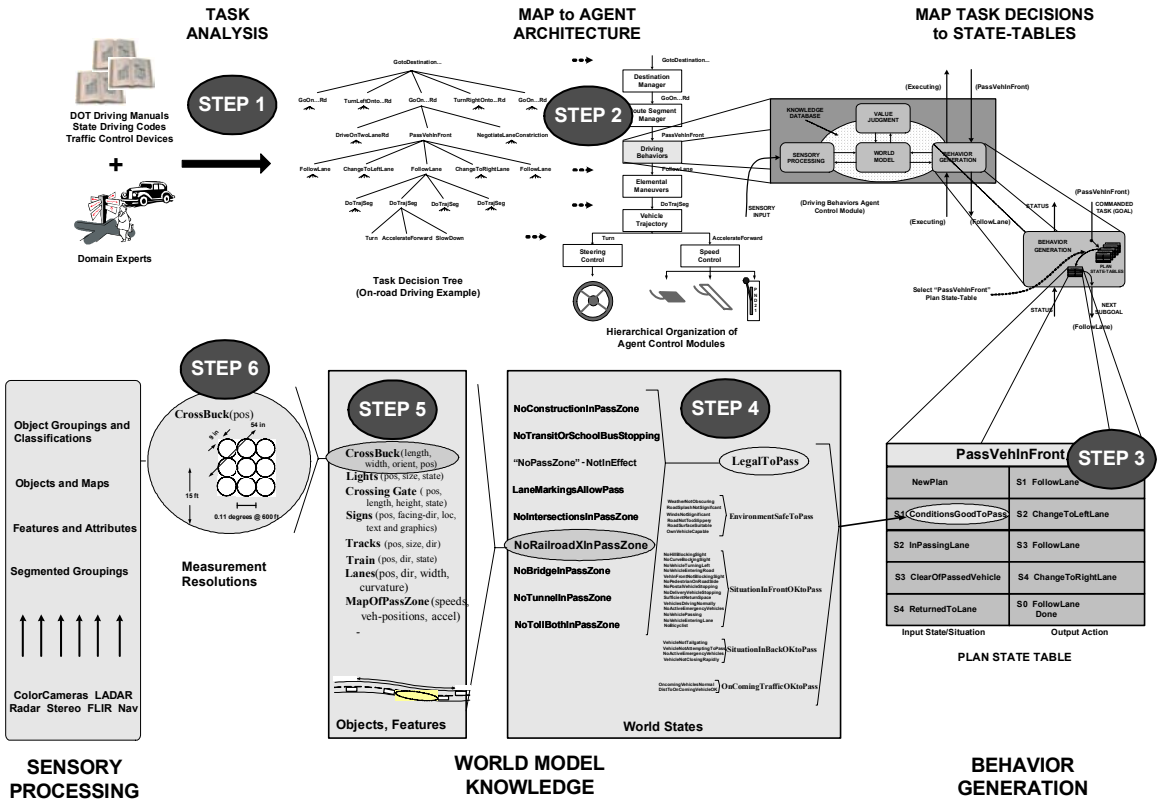

Figure 2. The six steps of the RCS methodology for knowledge acquisition and representation.

optimize a cost function that is defined by the global context of task goals, intent, priorities, and constraints.

5.0 Engineering Methodology

Although RCS is often thought of as a reference model architecture, as described above, there is also a corresponding methodology that accompanies it. Over the past thirty five years, as many different applications have been implemented using the RCS reference model architecture, an RCS software engineering methodology has evolved [11]. Thus, one can think of RCS as both an architecture and a methodology.

In Figure 2, an example of the RCS methodology for designing a control system for autonomous on-road driving under everyday traffic conditions is summarized in six steps. Each of these six steps is performed during design time, and is performed via human analysis (as described below).

5.1. Task Decomposition Design

The first step is to gather as much task related knowledge as possible with the goal of defining a set of commands that incorporate all of the activities at all levels of detail. For on-road driving this knowledge source would include driving manuals, state and federal driving codes, manuals on traffic control devices and detailed scenario narratives by subject matter experts (SMEs) of large numbers of different driving experiences.

Scenarios and examples are gone over in an attempt to come up with the names of commands that describe the activities at finer and finer resolutions of detail. Figure 3 provides an example. The high level goal of "Goto destination" (such as "go to post office") is broken down into a set of simpler commands – "GoOnRoad-name", "TurnLeft Onto-name" (MapQuest-like commands). At the next level down, these commands are broken down to simpler commands such as "Drive On Two Lane Road", "Pass Vehicle In Front" and these are then decomposed to yet simpler commands such as "FollowLane", ChangeToLeftLane", etc.

Four very important things are being done with the knowledge in this step.

1. The first is the discovery and naming of simpler component subtasks that go into making up the more complex tasks.
2. The second is that for each of these component subtasks, we are defining a subtask command.

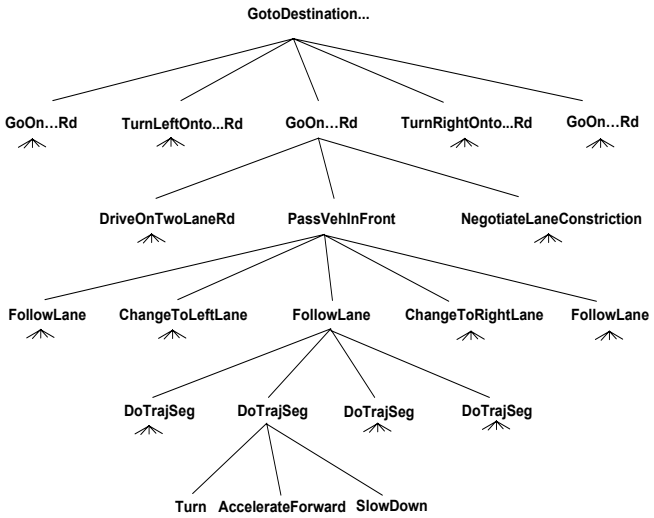

Figure 3. Task decomposition decision tree for on-road driving example.

3. The third is the understanding of the coordination of subtask activities that the task involves. This is identified by the analysis of scenarios of remembered specific examples.
4. The fourth is the careful grouping of these commands by layer and decomposition to ensure that the example on-road driving tasks can be completely described, from the start to finish of a scenario, by the proper sequencing of these commands at the appropriate levels.

This first step of the methodology sets the number of layers of agent control modules that will be required by step 2 (below) to execute the task decomposition.

5.2 Step 2 – Agent Control Module Organization

Once a set of commands is defined, we need an organization to execute them. This step is identical to laying out an organizational structure of people in a business or the military. You know what you want to do at various levels of detail – now you need an organization of intelligent agents to do it. This structure is built from the bottom up. The above detailed task decomposition will tell us how many layers of agents to have in our organization but not how many agents are at a level or how they are grouped and coordinated. This step starts at the bottom with an agent control module controlling each actuator in the system and then uses the knowledge of the task activities to understand which subordinate agents are grouped under which supervisor to best coordinate the task commands from Step 1.

Figure 4 illustrates how a grouping of agent control modules is assembled to accomplish the commands defined in Step 1. In this example, the lowest level servo control modules are represented by icons of the actuators being controlled. The steering servo control module is represented by a steering wheel icon, the brake servo by a brake pedal icon, etc. For this simple example, only four actuator control module icons are shown. The brake, throttle, and transmission servo agent control modules are grouped under a single supervisor agent control module, which we will call the Speed Control Agent. This supervisor agent control module will receive commands such as "AccelerateForward" (a more general form would be "Accelerate (magnitude, direction)) and have to coordinate its output commands to the brake, the throttle, and the transmission to accomplish them. By a similar analysis, the Steering Servo Agent is placed under a supervisor agent we call the Steering Control Agent Module. The Vehicle Trajectory Control Agent coordinates steering commands to the Steering Control Agent with the speed commands sent to the Speed Control Agent described above. The command decomposition of the commands at levels above the Vehicle Trajectory Control Agent are shown being executed by a single agent at each layer since there are no more subordinate agent control modules to be coordinated in this simple example. In a more realistic implementation, there would be additional agent control modules for ignition and engine starting, lights, turn signals, windshield wiper/washer, pan/tilt turrets that carry the sensor sets, etc. This would be the step at which the organizational structure would be defined to properly coordinate these modules' activities in accordance with step 1 as described in Section 5.1.

5.3. Step 3 – State-Table Definitions

At this stage of the knowledge definition process we know the vocabulary and syntax of commands. We also know what set of subcommands each command decomposes into, and where in the agent control hierarchy these decompositions take place. Step 3 is to establish the rules that govern each command's decomposition into its appropriate sequence of simpler output commands. These rules are discovered by first listing the approximate sequence set of output commands that correspond to a particular input command to an agent.

Figure 5 illustrates this step with a state-table of the "Pass Veh(icle) In Front" command at the Driving Behaviors Agent Control Module. This pass command is decomposed into five simpler output commands – "Follow Lane", "Change to Left Lane", "Follow Lane", "Change to Right Lane", and "Follow Lane" which are at the appropriate level of resolution for this layer in the agent hierarchy. These output commands can be read in sequence down the right hand column of the state table. The knowledge that is being added by this step 3 is to identify and name the situations (the left hand column of the state-table) that will transition the activity to each of these output commands. These named situations are the branching conditions that complete the task decision tree representation.

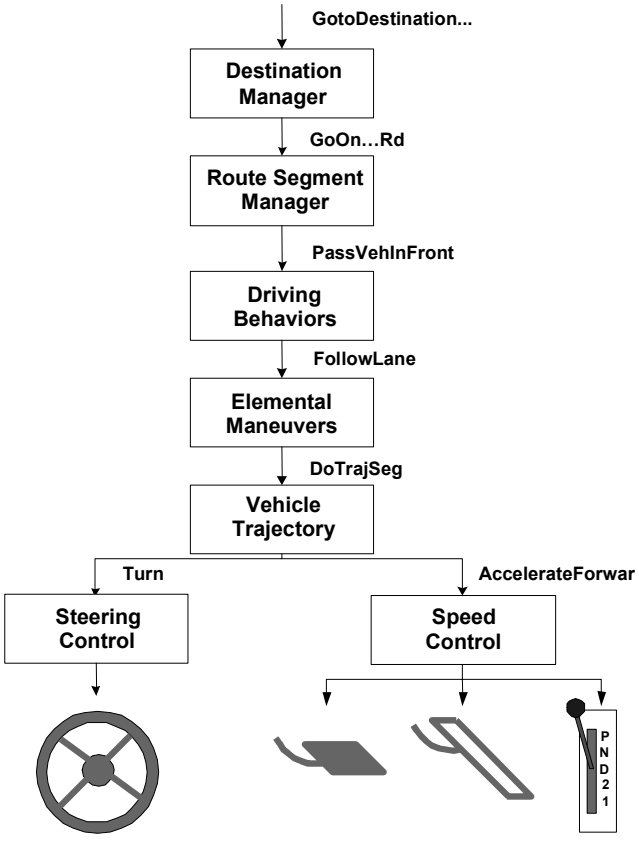

Figure 4. The hierarchical organization of agent control modules executing the task command decomposition.

Each of these newly named transition situations with their corresponding output command actions represent a single production rule that is represented as a single line in the state-table. The sequence in which these lines (rules) are executed is ordered by the addition of a state variable (S1, S2, etc). In the example in Figure 5, the first rule shown in the state-table says that if this is a "New Plan" (input condition), then the output action side of the rule (the right hand side of the state-table) sets the state to "S1" and outputs the command to "Follow Lane". As a result of executing this rule, this module is now in state "S1" and can only execute rules that include the state value of "S1" in their input condition. The only rules that will be searched by this module are those in the state-table that clusters the rules relating to this particular input command ("PassVehInFront"). In this state table, there is only one line (rule) that contains the state value "S1" as one of its input conditions. Thus, only that line can match and it will not match until the situation "ConditionsGoodToPass" is also true. When this situation occurs, this line will match (this rule will fire) and the module will go to state "S2" and output the command to "ChangeToLeftLane". This output command is sent to the subordinate agent control module (Elemental Maneuvers Control Module) where it becomes that module's input command invoking a corresponding state-table to be evaluated as described here.

PassVehInFront	
NewPlan	S1 FollowLane
S1 ConditionsGoodToPass	S2 ChangeToLeftLane
S2 InPassingLane	S3 FollowLane
S3 ClearOfPassedVehicle	S4 ChangeToRightLane
S4 ReturnedToLane	S0 FollowLane Done
Input State/Situation	Output Action

Figure 5. State-table for the Pass Vehicle In Front command.

Thus, the large set of rules governing the task decision tree execution is clustered both by the layer of resolution in the hierarchy and by the task context of the particular command at each layer so that only a very small number of rules have to be searched at any given time. The execution order of these selected rules is controlled by the addition of state values. It is important to note that the knowledge

discovery, representation and organization have been completely driven by looking at the problem from the detailed task decomposition point of view.

We will now make an important summary about the structuring of the knowledge base in these first three steps. These three steps were concerned with task knowledge expressed as the finer and finer branching of the decision process whereby the output action is sequenced in order to accomplish the assigned task. This third step identifies arbitrarily named situations we create to encompass everything that the task depends upon at this point and at this level of resolution in its execution. In this example, it was named "ConditionsGoodToPass".

These first three steps provide a complete listing of the task decomposition rules (i.e. these rules that determine when the system has to do something different in order maintain progress towards the goal.) These rules have been grouped into layers of resolution, and within each layer, clustered into tables of rules relevant to a single input command. Within each table they are ordered in their execution sequence by additional state values.

We can think of these first three steps as identifying the procedural knowledge involved in the task decomposition process, i.e. defining all of the task branching conditions and resultant corrective output actions. The next three steps are to identify all of the knowledge that is used to evaluate whether or not the branching conditions are true.

5.4. Step 4 – Situation Dependencies on World States

The world knowledge we want to identify and represent are those precursor world states that determine the task branching situation in the input side of the state-tables. This is best illustrated with an example. Figure 6 shows the "PassVehInFront" state-table. As discussed above, the output command to "Change To Left Lane" is issued when the "ConditionsGoodToPass" situation occurs. We ask of our expert knowledge sources "what do we have to know about the state of the world at this time to say that the conditions are good to pass". Again, we use detailed scenarios and our knowledge sources to drill down to the parameters that go into this situation assessment. We find that there is a surprisingly large set of things we want to know. We list these items as they come up in scenarios and from manuals – "there cannot be an on-coming car within the passing distance", "there must be a broken yellow lane marker on our side of center in the lane", "there cannot be a railroad crossing within the passing distance", "our own vehicle is not being passed", etc. We will call these items world states since they seem to describe certain attributes about the world that are relevant to our present task.

The purpose of this step is to provide a listing of all of the parameters (in terms of named world states) that affect whether the task branching condition situation is true or not. We have not identified the sensitivity of the final situation to these precursor values or what functions are used to weight and evaluate the individual or combined truth of these precursor values. The identification of these world states is independent of whatever technique is used to implement the control system. Different implementation paradigms will affect the sensitivity, weighting, costing, and other evaluation functions. For example, attributes like the level of driver aggressiveness may affect the calculation of the length of the required passing zone that is a precursor to a number of individual world state calculations related to the "ConditionsGoodToPass". How to best represent these functions and the variables they affect is still an area of research.

It should be noted that, although these conditions are determined at design-time, the execution of them are dependent upon parameters that are only made available at run-time. Two different implementations of a process described by the state machine can perform quite differently depending upon the parameters that are available at run-time.

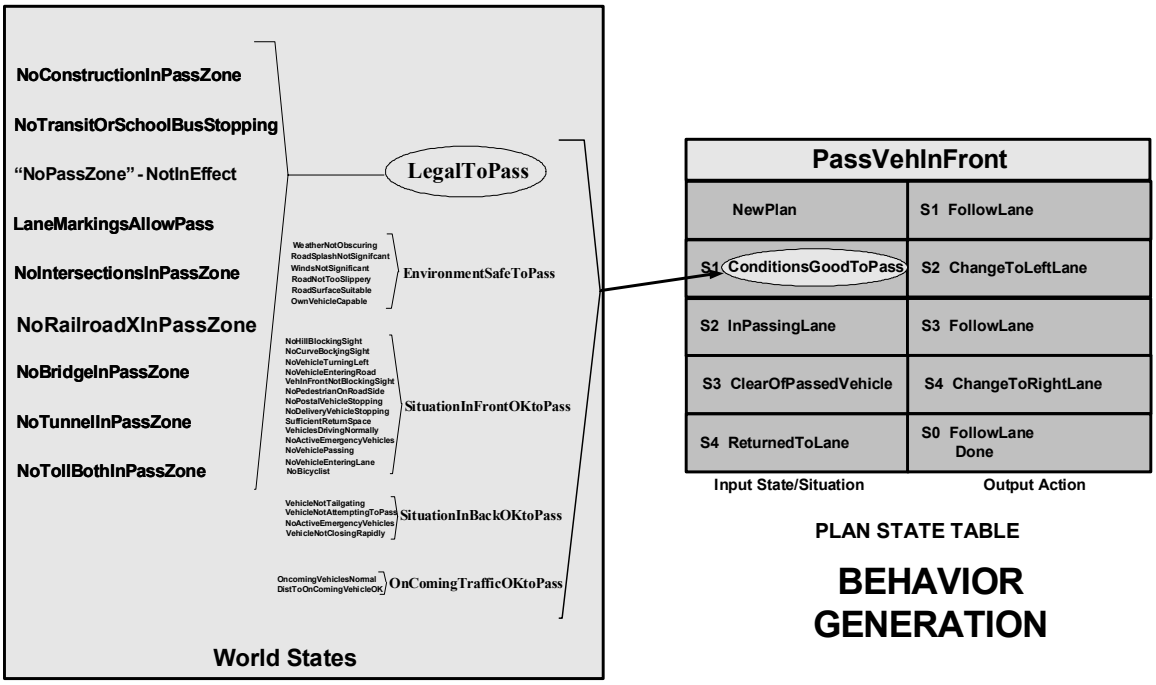

Figure 6. The identification of all of the precursor world states used to evaluate whether the situation "ConditionsGoodToPass" is true or not.

5.5. Step 5 – World State Dependencies on Objects

This step identifies all of the objects, their features and attributes that need to be measured by the sensing system to create the world model states described above. Figure 7 continues with the passing example. As described above, one of the aggregate world model states was "LegalToPass" which was a grouping of a number of related world states which all deal with various legal restrictions on the passing operation. One of these component world states that identify a legal restriction is "NoRailroadCrossingInPassZone". In this step, for each of the identified world states we wish to identify all of the objects, their features, and attributes relevant to creating each named world state value. For the world state named "NoRailroadCrossingInPassZone", these objects would include the railroad crossbuck emblem, crossing lights, crossing gate, crossing signs either alongside the road or painted on the road surface, the railroad tracks, and the train itself. For each of these objects, we identify characteristic features or attributes that will be used for recognition of the object (e.g. the width and length of the crossbuck planks) and/or its state (e.g. flashing lights or a lowered gate as indicator of active warning state).

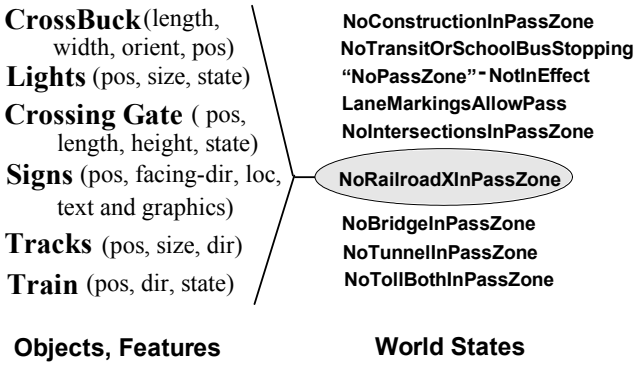

Figure 7. Example of the objects that are used to establish the "NoRailroadXinPassZone" world state.

Step 4 has defined a surprisingly large set of named world states that are relevant to decisions that have to be made at each decision point in the task. Now, in Step 5 we find that each of these world states might result from a number of objects and each of these has multiple features and attributes required for their recognition. We see that the size of the knowledge base is extremely large, but the RCS methodology's approach to the grouping and representation of this knowledge base has created a manageable structuring. We have a place to put each piece of knowledge and we use the task context to encode much of the relationship of the knowledge elements to each other.

5.6. Step 6 – Measurement Resolutions

In this last step, we want to define the resolution requirements for the measurement of objects for specific task decisions. We do this by determining the expected distances to these objects during particular task activities. In the case of the task activity of passing a vehicle in front,

we have to be able to see objects such as the railroad crossbuck at the far limit of the expected passing zone. For a vehicle passing on a 75 kph road, the passing zone could easily be 200 m or more. This means that the crossbuck, which is found at the railroad crossing itself, (whereas warning signs might be up to 300 m before the crossing) would have to be sensed and recognized by the sensory processing system at this distance. Since we know the size of the crossbuck plank elements, we can make an estimate of the absolute minimum sensory processing capability required to recognize it at this distance.

These specifications of the objects, attributes, features, and measurement resolutions have been derived from a detailed analysis of the world states required to evaluate a particular task branching condition situation. This allows us to provide a very detailed specification as to what sensory processing is required in order to do specific tasks and subtasks. This is important because one of the single biggest impediments to the implementation of autonomous driving control systems is the lack of capability of the sensory processing systems. The identification of the objects of interest for particular task activities focuses the attention of the sensory processing on these objects that should be measured at the present state of the task, leading to very efficient and effective use of this very compute intensive resource. It additionally helps to identify early on, during system design, which tasks are even feasible given the present state-of-the-art in sensory processing and points the direction to research areas to be developed for other capabilities to be realized.

This also allows for the development of performance specifications in order to qualify systems for different driving capabilities [10].

6.0 Conclusion and Future Prospects

In many ways, 4D/RCS is a superset of Soar, ACT-R, ICARUS, IMPRINT, Dickmanns 4-D approach [13], and even behaviorist architectures such as Subsumption [12] and its many derivatives. 4D/RCS incorporates and integrates many different and diverse concepts and approaches into a harmonious whole. It is hierarchical but distributed, deliberative yet reactive. It spans the space between the cognitive and reflexive, between planning and feedback control. It bridges the gap between spatial distances ranging from kilometers to millimeters, and between time intervals ranging from months to milliseconds. And it does so in small regular steps, each of which can be easily understood and readily accomplished through well known computational processes.

Each organizational unit in 4D/RCS refines tasks with about an order of magnitude increase in detail, and an order of magnitude decrease in scale, both in time and space. At the upper levels, most of the computational power is spent on cognitive tasks, such as analyzing the past, understanding the present, and planning for the future. At the lower levels, most of the computational power is spent in motor control, and the early stages of perception. 4D/RCS makes the processes of intelligent behavior understandable in terms of computational theory. Thus, it can be engineered into practical machines.

References

1. "SoarTech," *2004, http://www.soartech.com/htmlonly/projects.php*,.
2. Albus, J., "A Theory of Cerebellar Function," *Mathematical Biosciences*, Vol. 10, 1971, pp. 25-61.
3. Albus, J., "Outline for a Theory of Intelligence," *IEEE Transactions on Systems Man and Cybernetics*, Vol. 21, 1991, pp. 473-509.
4. Albus, J., "The NIST Real-time Control System (RCS): An Approach to Intelligent Systems Research," *Journal of Experimental and Theoretical Artificial Intelligence*, Vol. 9, 1997, pp. 157-174.
5. Albus, J. and Meystel, A., *Engineering of Mind*, John Wiley & Sons, Inc. 2001.
6. Albus, J., Schlenoff, C., Madhavan, R., Balakirsky, S., and Barbera, A., "Integrating Disparate Knowledge Representations Within 4D/RCS," *Proceedings of the 2004 AAAI "Achieving Human-Level Intelligence through Integrated Systems and Research" Fall Symposium*, 2004.
7. Albus, J. S., *Brain, Behavior, and Robotics*, McGraw-Hill 1981.
8. Anderson, J., *The Architecture of Cognition*, Lawrence Erlbaum Associates, Mahwah, N.J., 1983.
9. Archer, R., Lebriere, C., Warwick, W., and Schunk, D., "Integration of Task Network and Cognition Models to Support System Design," *Proceedings of the Collaborative Technology Alliances Conference: 2003 Advanced Decision Architectures*, College Park, MD, 2003.
10. Barbera, A., Horst, J., Schlenoff, C., Wallace, E., and Aha, D., "Developing World Model Data Specifications as Metrics for Sensory Processing for On-Road Driving Tasks," *Proceedings of the 2003 Performance Metrics for Autonomous Systems (PerMIS) workshop*, 2003.
11. Barbera, T., Fitzgerald, M., Albus, J., and Haynes, L. S., "RCS: The NBS Real-Time Control System," *Proceedings of the Robots Conference and Exposition*, Detroit, Michigan, 1984.
12. Brooks, R. A., "A Robust Layered Control System for a Mobile Robot," MIT AI Lab, A. I. Memo 864, Sept. 1985.
13. Dickmanns, E. D., "An Expectation-Based Multi-Fical Saccadic (EMS) Vision System for Vehicle

Guidance," *Proceedings of the 9th International Symposium on Robotics Research (ISRR'99)*, Salt Lake City, 1999.
14. Fodor, J., *The Mind Doesn't Work That Way*, MIT Press, Cambridge, MA, 2000.
15. Gonzalez, C., "ACT-R Implementation of an Instance-Based Decision Making Theory," *Proceedings of the Collaborative Technology Alliance Conference: 2003 Advanced Decision Architectures*, College Park, MD, 2003.
16. Kieras, D. and Meyer, D. E., "An Overview of the EPIC Architecture for Cognition and Performance With Application to Human-Computer Interaction," *Human-Computer Interaction*, Vol. 12, 1997, pp. 391-438.
17. Kurzweil, R., *The Age of Spiritual Machines*, Penguin Books, New York, 1999.
18. Laird, J. E., Newell, A., and Rosenbloom, P. S., "SOAR: An Architecture for General Intelligence," *Artificial Intelligence*, Vol. 33, 1987, pp. 1-64.
19. Minton, S. N., "Quantitative Results Concerning the Utility of Explanation-Based Learning," *Artificial Intelligence*, Vol. 42, 1990, pp. 363-391.
20. Pierce, C. S., *Collected Papers, Band VII (Hrsg.)*, Arthur W. Burks 1958.
21. Pylyshyn, Z., *The Robot's Dilemma: The Frame Problem in Artificial Intelligence*, Ablex, Norwood, N.J., 1987.
22. Searle, J., *The Rediscovery of the Mind*, MIT Press, Cambridge, MA, 1992.
23. Shapiro, D. and Langley, P., "Controlling Physical Agents Through Reactive Logic Programming," *Proceedings of the Third International Conference on Autonomous Agents 386-387*, ACM Press, Seattle, 1999.
24. Shoemaker, C., Bornstein, J., Myers, S., and Brendle, B., "Demo III: Department of Defense Testbed for Unmanned Ground Mobility," *SPIE Conference on Unmanned Ground Vehicle Technology, SPIE Vol. 3693*, Orlando, FL, 1999.