Available online at www.sciencedirect.com
SCIENCE DIRECT®
Robotics and Autonomous Systems 49 (2004) 67–78
Robotics and Autonomous Systems
www.elsevier.com/locate/robot

How task analysis can be used to derive and organize the knowledge for the control of autonomous vehicles

T. Barbera*, J. Albus, E. Messina, C. Schlenoff, J. Horst

National Institute of Standards and Technology, Intelligent Systems Division, 100 Bureau Drive, Gaithersburg, MD 20899, USA

Received 27 July 2004; accepted 27 July 2004

Abstract

The real-time control system (RCS) methodology has evolved over a number of years as a technique to capture task knowledge and organize it in a framework conducive to implementation in computer control systems. The fundamental premise of this methodology is that the present state of the task activities sets the context that identifies the requirements for all the support processing. In particular, the task context at any time determines what is to be sensed in the world, what world model states are to be evaluated, which situations are to be analyzed, what plans should be invoked, and which behavior generation knowledge is to be accessed. This results in a methodology that concentrates first and foremost on the task definition. It starts with the definition of the task knowledge in the form of a decision tree that clearly represents the branching of tasks into layers of simpler and simpler subtask sequences. This task decomposition framework is then used to guide the search for and to emplace all of the additional knowledge. This paper explores this process in some detail, showing how this knowledge is represented in a task context-sensitive relationship that supports the very complex real-time processing the computer control systems will have to do.

Keywords: 4D/RCS methodology; Autonomous vehicle; Task analysis; Control system; Task decomposition

1. Introduction

One of the challenges of building complex control software is the need to capture a human's knowledge and translate it into a form that is executable by a computer. The capture and translation processes are fraught with possibilities for losing knowledge and introducing error. Typically, once a system is implemented, the software's decision process and internal knowledge is not directly recognizable by the human expert, making it difficult to assess whether the system was correctly implemented and provide enhancements to the system. Furthermore, there are very few guidelines available to implementers as to exactly which knowledge must be

* Corresponding author. Tel.: +1 301 975 3460; fax: +1 301 990 9688.
E-mail addresses: barbera@nist.gov, tony.barbera@nist.gov (T. Barbera), james.albus@nist.gov (J. Albus), elena.messina@nist.gov (E. Messina), craig.schlenoff@nist.gov (C. Schlenoff), john.horst@nist.gov (J. Horst).

0921-8890/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.robot.2004.07.017

included, where to place it within the system, and how to represent it.

The RCS methodology and hierarchical task decomposition architecture has been used to implement a number of diverse intelligent control systems. An application summary [1] describes major systems, including machining stations, space robotics, coal mining, and stamp distribution systems. Most recently, experimental validation of the 4D/RCS architecture has been provided by the performance of the Demo III experimental unmanned ground vehicles (XUVs) in an extended series of demonstrations and field tests during the winter of 2002–2003.

During three major experiments designed to determine the technology readiness of autonomous driving, the Demo III experimental unmanned vehicles were driven a total of 550 km, over rough terrain: (1) in the desert; (2) in the woods, through rolling fields of weeds and tall grass, and on dirt roads and trails; and (3) through an urban environment with narrow streets cluttered with parked cars, dumpsters, culverts, telephone poles, and manikins. Tests were conducted under various conditions including night, day, clear weather, rain, and falling snow. The unmanned vehicles operated over 90% of both time and distance without any operator assistance. A detailed report of these experiments has been published [2].

It should be noted that the Demo III tests were performed in environments devoid of moving objects such as on-coming traffic, pedestrians, or other vehicles. The inclusion of moving objects in the world model, and the development of perception, world modeling, and planning algorithms for operating in the presence of moving objects is a topic of current research.

2. Background

An RCS system models complex real-time control as three major processing components (Fig. 1):

(1) sensory processing to measure, recognize, and classify entities and events of task interest in the environment and place them into an internal world model;
(2) internal world model processing that represents and derives world states, situations, and evaluations in a task context manner; and
(3) behavior generation processing that reasons from this world model, selects plans, and makes value judgments to decide on the next appropriate output action to accomplish the mission goals.

These three components work together, receiving a task, breaking it down into simpler subtasks, determining what has to be known in the internal world model to decide on the next course of action, and alerting the sensory processing as to what internal world objects have to have their states updated by new sensory readings. All together, this produces

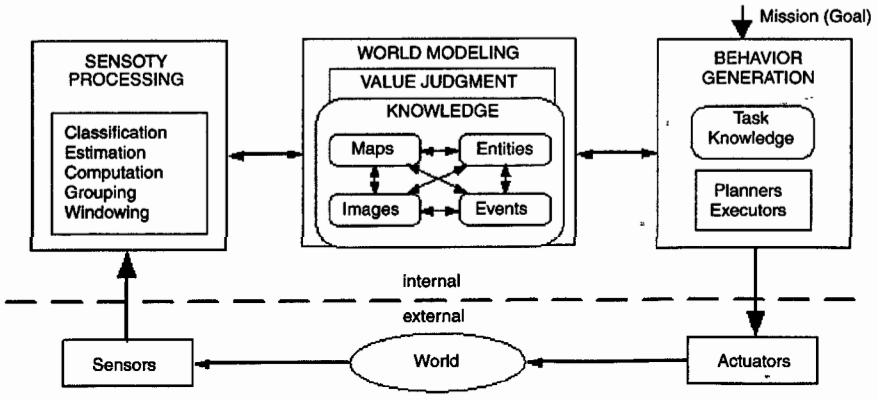


Fig. 1. The basic internal structure of a 4D/RCS control loop. Sensory processing performs the functions of windowing, grouping, computation, estimation, and classification on input from sensors. World modeling maintains knowledge in the form of images, maps, entities, and events with states, attributes, and values. Value judgment provides criteria for decision-making. Behavior generation is responsible for planning and execution of behaviors.

goal-directed, sensory-interactive, adaptive, stable, real-time accomplishment of the input goal.

A large number of complex real-time control systems have been built at the National Institute of Standards and Technology (NIST) and other research organizations using the NIST-defined real-time control system (RCS, most recently referred to as 4D/RCS) design methodology and reference architecture [3]. These systems have as their backbone a hierarchical organization of agent control modules, each of which does a partial task decomposition of its input goal task, and outputs simpler subtask goals to the next lower subordinate agent control module. Each of these agent control modules is made up of the three major processing components of sensory processing, world modeling, and behavior generation. Each agent control module is concerned with only its own level of responsibility in the decomposition of the task.

This paper will describe the RCS methodology in its application to the task of autonomous on-road driving to illustrate its approach to "mining" and representing task knowledge for control system implementation.

3. Representation and methodology comparison

To better understand the approach that 4D/RCS uses to represent knowledge, it is important to understand how it is different than other existing architectures.

While our understanding of how the human mind represents, stores, and retrieves various classes of knowledge is limited, considerable research has been performed in developing computer architectures to support these operations for autonomous systems. The architectures are commonly grouped into the classes of deliberative and reactive [4], with most architectures lying somewhere in-between these two extremes. Reactive architectures strive to embed the control strategy into a collection of pre-programmed reactions (sense–action mappings) that are very similar to human reflexes [5]. This approach provides a direct, constant-time response to the sensed environment, which requires an expert to isolate each possible combination of sensor output and map them to actions. Adding the equivalent of the subconscious' sensory and procedural skill memory to the above architecture presents the basis for the behavior-based architectures [6].

Behavior-based systems may implement very sophisticated control laws and include the use of both semantic and episodic long-term memory. These systems do not use an explicit world model and behaviors are activated based on environmental input, making them similar to the human subconscious which is capable of sophisticated actions without explicit conscious control. Typically, behaviors are implemented in a layered structure with the lowest layer being constructed of relatively simple, self-contained control laws that are more time-extended than their reactive cousins [7] and which utilize short-term memory.

The addition of an explicit world model and the ability to simulate and reason over the consequences of intended actions formulates the basis for the class of deliberative architectures. In systems such as the one in [8], a multitude of possible system actions are explored and a conscious decision is made that is based on the cost/benefit of each action chain. It is the authors' belief that an architecture that allows such conscious decisions to be made is a requirement of the development of intelligent systems. While this paper focuses on the RCS architecture, various other deliberative architectures such as CIRCA [9] and the three-level intelligent machine [10] may be found in the literature. RCS is unique in its inherent reuse of architectural components and knowledge representations.

From a knowledge acquisition (KA) perspective, RCS is not formal, like EXPECT [11], CommonKADS [12], and other methods and tools. These other approaches focus more on the knowledge in the system by knowledge engineers, rather than the construction of an executable system that is accessible to the domain expert. Their application domains also tend to be aimed at business and other process modeling, instead of real-time control for autonomous systems. RCS blends the knowledge acquisition process with classical control system design.

4. RCS methodology summary

The RCS methodology concentrates on the task decomposition as the primary means of understanding the knowledge required for intelligent control. This approach begins with the knowledge "mining" activities to retrieve knowledge from subject matter experts (SMEs). The gathering and formatting of this knowl-

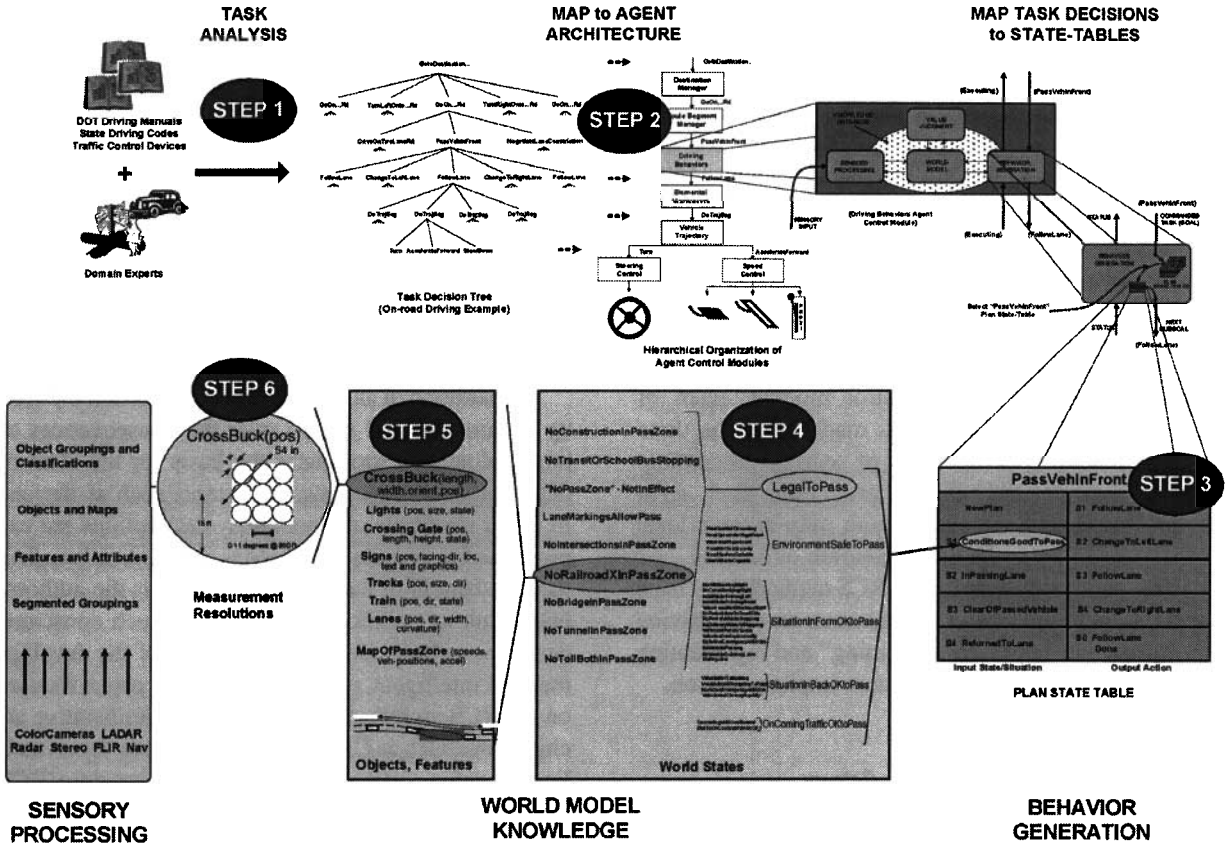


Fig. 2. The six steps of the RCS methodology approach to knowledge acquisition and representation.

edge can be summarized in six steps, each of which follows from the knowledge uncovered by the previous steps (Fig. 2 presents a high-level summary view of the overall approach):

(1) The first step involves an intensive analysis of domain knowledge from manuals and subject matter experts, especially using scenarios of particular subtask operations. The output of the effort is a structuring of this knowledge into a task decision tree form of simpler and simpler commands (actions/verbs) at simpler and simpler levels of task description.
(2) This step defines the hierarchical organization of agent control modules that will execute these layers of commands in such a manner as to reasonably accomplish the tasks. This is the same as coming up with a business or military organizational structure of agent control modules (people, soldiers) to accomplish the desired tasks. This step forces a more formal structuring of all the subtask activities as well as defining the execution structure.
(3) This step clarifies the processing of each agent's input command through the use of rules to identify all the task branching conditions with their corresponding output commands. Each of these command decompositions at each agent control module will be represented in the form of a state-table of ordered production rules (which is an implementation of an extended finite state machine (FSM)). The sequence of simpler output commands required to accomplish the input command and the named situations (branching conditions) that transition the state-table to the next output command is the primary knowledge represented in this step.
(4) In this step, the above-named situations that are the task branching conditions are defined in great detail in terms of their dependencies on world and

task states. This step attempts to define the detailed precursor states of the world that cause a particular situation to be true.

(5) In this step, we identify and name all the objects and entities together with their particular features and attributes that are relevant to defining the above world states and situations.

(6) The last step is to use the context of the particular task activities to establish the distances and, therefore, the resolutions at which the above objects and entities must be measured and recognized by the sensory processing component. This step establishes a set of requirements and/or specifications for the sensor system at the level of each separate subtask activity.

We will now cover these six steps in detail using the on-road driving example.

4.1. Step 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 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 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. Fig. 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/name.

(3) The third is the understanding of the coordination of subtask activities that the task involves. This is

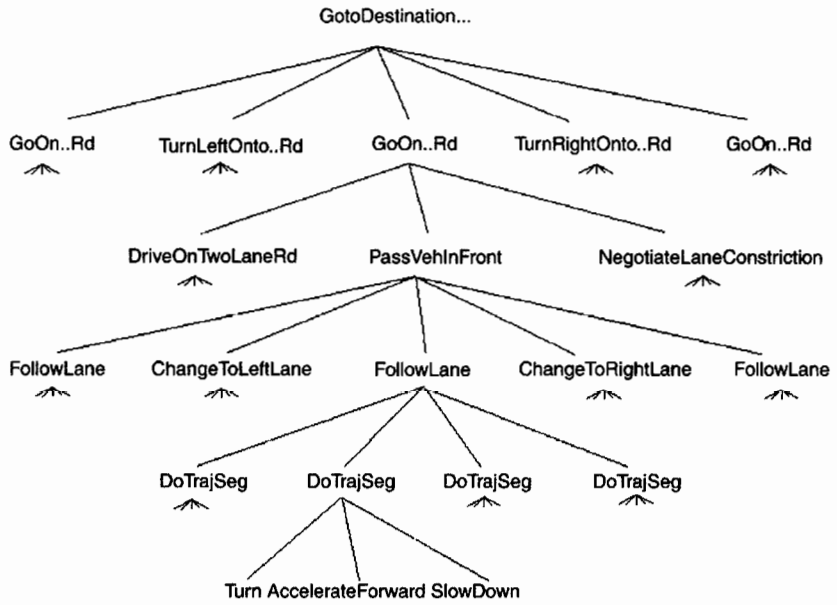


Fig. 3. Task decomposition tree for on-road driving example which shows the simpler commands that are used at each lower layer to represent the finer and finer resolutions of detail activities.

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 (step 2) to execute the task decomposition.

4.2. 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. Here, we decide which agent control modules will execute which subtasks. 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.

Fig. 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 "Accelerate-Forward" (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.

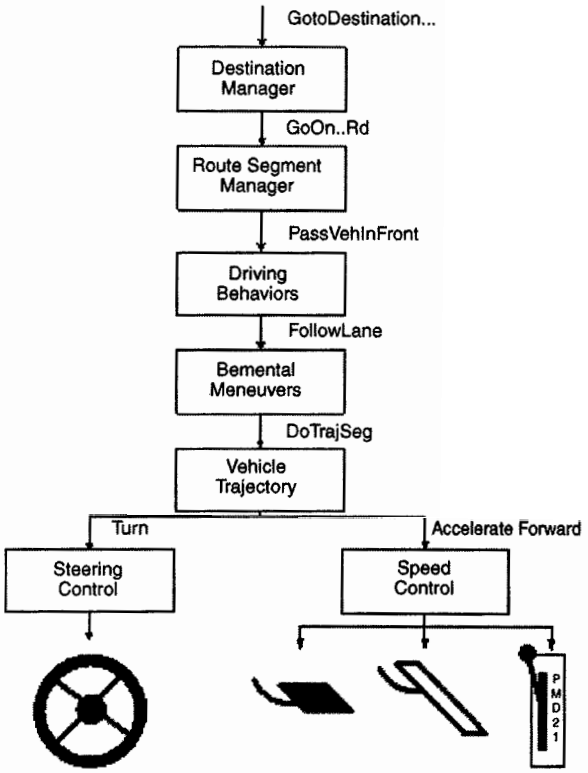


Fig. 4. The hierarchical organization of agent control modules that are to execute the task command decomposition.

4.3. Step 3 – state-table definitions

At this stage of the knowledge definition process, we know the vocabulary and syntax of commands. We

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

Fig. 5. State-table for the Pass Vehicle In Front command. Each line is a rule relevant to this task. The left column contains the state values used for ordering the execution and the situations that specify the branching conditions of the task decision tree. The right column contains the output commands.

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 and coordination 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.

Fig. 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.

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 that these lines (rules) are executed is ordered by the addition of a state variable (S1, S2, etc). In the example in Fig. 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. 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.

More complex state-tables are easily built that can represent task decompositions that are not only sequential but include parallel coordination of multiple subordinates, as well as multiple possible next actions based on which particular branching situations occur in real-time. These are easily represented by the use of ordering state variables and the specification of the discriminating situations.

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 task decomposition process, the definition of particular agents to carry this out, and the identification of

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, one of these situations 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 single input commands. Within each table they are ordered in their execution sequence by additional state values. It is important to note that although these rules are used in a state-table format throughout this paper, this is not the only way they can be applied. Exploration is currently ongoing to apply these rules to a graph-based planning systems [5] as a mechanism to determine the states in the world that costs need to be applied to.

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.

4.4. 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. Fig. 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 then determine what we have to know about the state of the world at this time to say that the conditions are good to pass. Using detailed scenarios and our knowledge sources, we determine the parameters that go into this situation assessment. This process makes visible the surprisingly large set of things we want to know. These items are documented 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. These items are referred to as world states since they seem to describe certain attributes about the world that are relevant to our present task.

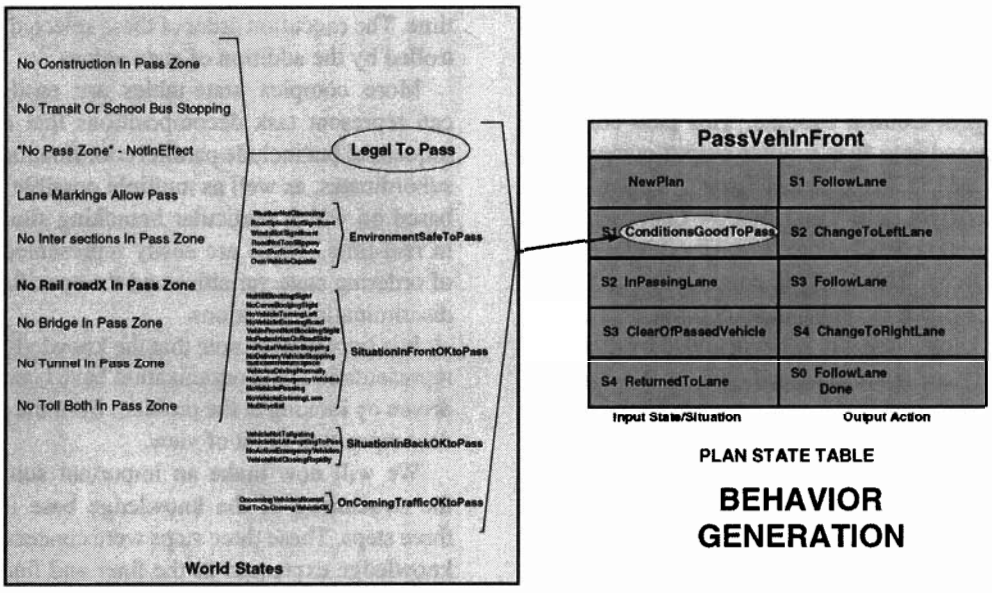


Fig. 6. The identification of all the precursor world states used to evaluate whether the situation "ConditionsGoodToPass" is true or not.

Once we have listed the world states relevant to the situation of "ConditionsGoodToPass", we go over the list and see if groupings can be made. In this case, we group some world states into a category we call "LegalToPass", and others into additional categories we call "EnvironmentSafeToPass", "SituationInFrontOkToPass", "SituationInBackOkToPass", "OncomingTrafficOkToPass", etc. These groupings are aggregate world states leading to the situation "ConditionsGoodToPass". For this situation to be true, all the aggregate world states have to be true. For each of the aggregate world states to be true, all their component world states have to be true.

The purpose of this step is to provide a listing of all the parameters (in terms of named world states) that affect whether the task branching condition situation is true or not. It is important to note that 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. Their evaluation functions could be an AND/OR tree or more complex relationships but at this stage we only want to identify the list of parameters. It is an additional part of the implementation process that will decide on the functional relationships to use.

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 aggressivity 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.

4.5. Step 5 – world state dependencies on objects

This step identifies all the objects, their features, and attributes that need to be measured by the sensing system to create the world model states described above. Fig. 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.

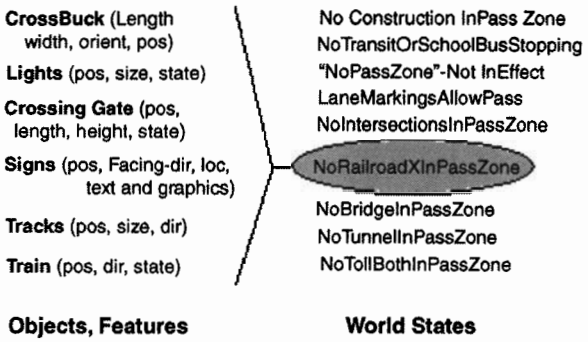


Fig. 7. Example of the objects that are used to establish the "NoRailroadXinPassZone" world state.

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 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 cross-buck 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 cross-buck planks) and/or its state (e.g. flashing lights or a lowered gate as indicator of active warning 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 are starting to get an idea of the size of the knowledge base, and it is extremely large. The good news is that 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. This structuring also provides an excellent indexed structure to easily find where to add knowledge as it is discovered, leading to an evolvable system, which

is possibly the most important system feature of this technique.

4.6. Step 6 – measurement resolution

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 cross-buck 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 cross-buck, which is found at the railroad crossing itself, (whereas warning signs might be 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 cross-buck plank elements, we can make an estimate of the absolute minimum sensory processing capability required to recognize it at this distance. This works out to sensors having a minimum resolution capability of 0.07°.

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 [13].

5. Summary and conclusion

This paper has presented a description of the use of the task-decomposition-oriented RCS methodology as an approach to acquiring and structuring the knowledge required for the implementation of real-time complex control tasks. Some form of structuring is essential to this process because complex control tasks such as on-road autonomous driving are characterized by extremely large knowledge sets that would be impossible to deal with if not for some way to organize them. The task decomposition approach to this structuring has given us a single consistent search process of well-defined sequential steps to both discover the relevant knowledge and to organize it. The knowledge has been partitioned into two large elements:

(1) task knowledge (i.e. how to do things) concerned with the description and representation of the task decomposition through layers of agent control modules performing control decision branching decisions, encoded as rules in state-tables;
(2) world knowledge (i.e. what is the situation) concerned with the description and representation of all the states of the world that are used to generate each of these condition branching situations in each state-table at each layer of the agent control hierarchy.

The organization of the knowledge has been driven by a requirement to identify everything according to its relevance to task activities. This has a very important impact on the implementation. This organization of the task knowledge is in a form that can be directly implemented. It has partitioned the knowledge by layers of resolution or abstraction and so high-level commands are clustered higher in the hierarchy and low-level, equipment control knowledge is clustered at lower levels in the hierarchy. It has threaded access to all of the knowledge from the task through world model states to objects and their attributes to be measured. This is exactly the form the control system needs, so it can access all the information relevant to the present task activity as rapidly as possible.

This organization of the knowledge leaves the knowledge in a form that continues to be easily accessible by non-programmer SMEs because all the knowledge is indexed through the task behavior

which is in exactly the same format that the SMEs remembered it for example scenario discussions. This means that continual evolution and updating of the knowledge structure is possible by both the SMEs and the system designers, together or separately. New task commands, new branching condition situations and output commands, new world states, additional objects, features and attributes are all easily added and the appropriate place to add them is indexed by the task decomposition itself.

In the case of on-road driving, the amount of knowledge that is needed to be captured via the approach described in this paper is large but not infinite. It is estimated that to perform intelligent on-road driving, 129 state-tables would be necessary containing 1000 situations, 10,000 world model states, 1000 world model entities, 7000 world model attributes. Details behind how these numbers were determined can be found in [14].

The research challenge is to develop a computer-based knowledge storage mechanism to capture the results of this process which up to now has been put into computers as a combination of drawings, word documents, spread sheets, and databases. The hope is that ontology tools and techniques will provide a more consistent single representational solution to capturing this knowledge and all the implied relationships, especially to the task, in a more computer readable and processible form.

6. Product disclaimer

The identification of certain commercial products or companies does not imply recommendations or endorsements by NIST.

Acknowledgement

This work was supported in part by the Army Research Lab's program in unmanned vehicles (PM. C. Shoemaker) and DARPA's Mobile Autonomous Robotic Software program (PM. D. Gage).

References

[1] J. Albus, The NIST real-time control system (RCS): an application survey, in: Proceedings of the 1995 AAAI Spring Symposium, 1995.

[2] R. Camden, B. Bodt, S. Schipani, J. Bornstein, R. Phelps, T. Runyon, F. French, C. Shoemaker, Autonomous mobility technology assessment: interim report, Technical Report ARL-MR-565, Army Research Laboratory, ATTN: AMSRL-WM-RP, Aberdeen Proving Ground, MD, 2003.

[3] J. Albus, A. Meystel, Engineering of Mind, Wiley, 2001.

[4] R.C. Arkin, Towards the unification of navigational planning and reactive control, in: Proceedings of the AAAI 1989 Spring Symposium on Robot Navigation, 1989.

[5] S. Balakirsky, A Framework for Planning with Incrementally Created Graphs in Attributed Problem Spaces, Akademische Verlagsgesellschaft Aka GmbH, Berlin, 2003.

[6] R. Brooks, Achieving Artificial Intelligence through Building Robots, Massachusetts Institute of Technology, 1986.

[7] M. Mataric, Behavior-based control: examples from navigation, learning, and group behavior, J. Exp. Theor. Artif. Intell. 9 (2–3) (1997) 323–336.

[8] A. Lacaze, Y. Moscovitz, N. DeClaris, K. Murphy, Path planning for autonomous vehicles driving over rough terrain, in: Proceedings of the ISIC/CIRA/ISAS'98 Conference, 1998.

[9] D. Musliner, E. Durfee, K. Shin, CIRCA: a cooperative intelligent real-time control architecture, IEEE Trans. Syst. Man Cyber. 23 (6) (1993) 1561–1574.

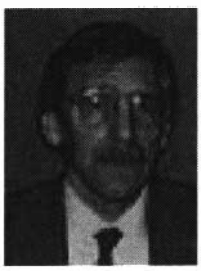
[10] G. Saridis, On the theory of intelligent machines: a survey, in: Proceedings of the 27th IEEE Conference on Decision and Control, 1988, pp. 1799–1804.

[11] B. Swartout, Y. Gill, Flexible knowledge acquisition through explicit representation of knowledge roles, in: Proceedings of the 1996 AAAI Spring Symposium on Acquisition, Learning, 965 and Demonstration: Automating Tasks for Users, 1996.

[12] A. Schreiber, B. Wielinga, R. DeHood, J. Akkermans, W.V. de Velde, CommonKADS: a comprehensive methodology for KBS development, IEEE Expert 9 (6) (1994) 28–37.

[13] T. Barbera, J. Horst, C. Schlenoff, E. Wallace, D. Aha, Developing world model data specifications as metrics for sensory processing for on-road driving tasks, in: Proceedings of the 2003 PerMIS Workshop, NIST Special Publication 990, Gaithersburg, MD, 2003.

[14] J. Albus, J. Evans, C. Schlenoff, T. Barbera, E. Messina, S. Balakirsky, Achieving intelligent performance in autonomous driving, Technical Report, National Institute of Standards and Technology (NIST) Internal Report, 2003.

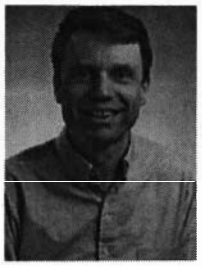


T. Barbera is a senior controls engineer in the Intelligent Systems Division at the National Institute of Standards and Technology (NIST), with 30 years of experience implementing real-time control systems using the 4D/RCS methodology.

J. Albus is a senior technical fellow in the Intelligent System Division at the National Institute of Standards and Technology. He is the co-developer of the NIST real-time control system architecture and has been involved in the study of intelligent systems for over 30 years.

E. Messina is knowledge systems group leader in the Intelligent Systems Division at NIST. She has over 20 years of experience in robotics and in software engineering for complex systems.

C. Schlenoff has a bachelors degree from the University of Maryland and a masters degree from Rensselaer Polytechnic Institute in mechanical engineering. He is a researcher in the Intelligent Systems Division at NIST, focusing on knowledge representation applied to autonomous systems and manufacturing. He was recently the Process Engineering Program Manager at NIST as well as the Director of Ontologies at VerticalNet.



J. Horst is with the Intelligent Systems Division of the National Institute of Standards and Technology where he is currently doing research measuring system intelligence, building on-road autonomous vehicle controllers, and designing conformance tests for implementations of standard interface languages. He received a BA (music) in 1976, a BSEE in 1985, and a MSEE in 1991 all from the University of Maryland, College Park.