Software Engineering for Intelligent Control Systems

Anthony Barbera, Elena Messina, Hui-Min Huang, Craig Schlenoff, Stephen Balakirsky

Understanding how humans manage information and carry out actions provides a foundation for the software engineering of intelligent control systems. The Real-time Control System (RCS) design methodology and reference architecture, founded on human information management concepts, provides a domain independent approach to the acquisition and representation of expert knowledge and supports implementation in a form conducive to maintenance of the knowledge base. Its architecture provides a structured framework for sensory, semantic, episodic, and procedural knowledge.

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 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 [Albus 1995] describes major systems, including machining stations, space robotics, coal mining, and stamp distribution systems. The most advanced RCS system is presently under development as the autonomous control system of the Army's Demo III unmanned vehicle program [Lacaze et.al. 2002]. This project's goals [Albus and Meystel 2001] are to create near-human levels of performance for vehicle driving and to be able to carry out military tactical maneuvers.

Understanding of how the human mind functions is slowly developing. Various models of the mind partition its activities in a number of ways. One model proposes a conscious and subconscious mind. The conscious mind reasons, plans activities, and sets goals. The subconscious mind handles all of our sensory inputs and controls autonomic functions.

Another way to think of the mind is to model how its memory works. Memory is categorized into subconscious and conscious. The subconscious memory stores all the patterns that sensory processing uses to match inputs in order to recognize objects in the world, sense temperatures or collisions, etc, and to execute procedural skills, such as controlling vocal cords to speak. The manner in which the subconscious functions is too poorly understood to be of use to designers of real-time control systems. More is known about conscious memory, which is generally modeled with two components: short term, and long term. Short-term memory is like a scratch pad for information being processed. It decays rapidly and has a very limited capacity. This limitation has manifested itself by forcing humans to "chunk" information to make sure the typical limit of 7±2 elements is not exceeded [Miller 1956]. An additional characteristic of this limitation is the use of analogies. We try to find what is similar so we can learn the base information content once and recognize how similar things are like it. [Dunbar and Schunn 1990]

Studies of long-term memory have classified two types: semantic and episodic [Tulving 1972]. The semantic long-term memory is a structured record of facts, objects and relations between objects [Atkinson and Shiffrin 1968]. Information is moved from short-term memory to this long-term semantic memory through repetition. We retrieve information from semantic memory through association of semantically related properties, e.g., the number 1492 recalls the trip to America by Columbus. The episodic long-term memory is our memory of events and experiences in the sequence that they occurred [Tulving 1984].

Models for memory and retrieval mechanisms are important for two reasons. First, an understanding of how humans retrieve stored knowledge helps better "mine" the expert for the particular task knowledge of the system. Second, an understanding of how we manage and store information will help design a better framework to hang this knowledge on so system designers can access and manipulate it.

2 Representational Architecture

A domain expert's knowledge is stored in episodic memory as sequences of events and operations. On-road driving will be used as an example. The driving task can be characterized by "steering the car to stay on the road and coordinate with the gas pedal and brake pedal to keep from running into things." An expert has done this in a large number of different situations and has an extremely large database of experience of different circumstances and corresponding response activities to successfully deal with them. The expert has derived semantic knowledge such as "wet streets are slippery and driving speed must be reduced."

The expert's knowledge is stored in the conscious and subconscious long-term memory. The subconscious mind has the sensory patterns stored to recognize entities, such as traffic signs, and the procedural skills to coordinate the steering wheel with the gas pedal to move the vehicle in the manner that the conscious mind has determined. The domain expert's knowledge that is most useful is from the conscious

long-term episodic and semantic memory. For the on-road driving example, the expert is interrogated to recall the semantic memory components i.e., the rules-of-the-road to establish the legal driving operations. The expert's episodic memory (detailed sequential task knowledge) is obtained through the expert's retelling of different scenarios. A particularly important component of this episodic knowledge is the recall of the coordination activities involved in carrying out various tasks. This is contrasted with most other control system design techniques where the knowledge is captured as functional (semantic) with little if any scenario-based (episodic) knowledge. Failure to account for the coordination is a major malfunction cause during integration.

The RCS methodology uses a hierarchical task decomposition representational format on which to hang the domain knowledge. Hierarchies are the architectural mechanisms used to "chunk" and abstract information to meet the mind's limitation of only being able to process 7±2 elements at one time. The scenario descriptions of intelligent control system activities naturally evolve into a task decomposition representation since the scenarios are task sequences and can easily be discussed at many levels of abstraction leading to well-defined levels within the task hierarchy. This hierarchy also acts as a convenient mechanism to place the semantic knowledge from the expert. Since each layer in the task decomposition represents a different level of abstraction of the tasks, it also delineates the context for incorporation of semantic knowledge relevant to that level of detail of the task's activities (Figure 1). A detailed description of this process can be found in [Huang et.al. 2001a]

3 Implementation

A correct implementation of an intelligent system must reliably perform a set of tasks and be able to accommodate augmentation of its capabilities. These requirements depend on how understandable the system is and thus how well it complies with the human limitations in information management. The key aspects of the RCS software engineering methodology are:

Hierarchical architecture of agent control modules. Using a task decomposition decision hierarchy to capture the knowledge from the expert's narratives helps instantiate this into an implementation of a hierarchical architecture of agent control modules (Figure 1) executing this task decomposition in a one-to-one fashion. This technique maintains the layered partitioning of the task to create levels of abstraction, task responsibility, execution authority, and knowledge representation in a manner so as to greatly enhance the designer's ability to think about each of these layers separately. Modifications and enhancements to this layer can be evaluated with respect to their completeness and potential interaction with other task activities at that same level of abstraction.

A generic agent control module (Figure 2) was developed as the unit building block in the hierarchical execution system. Each agent control module receives an input task command and decomposes it into a simpler set of commands to send to its subordinate(s), mimicking a chain-of-command organization. The input task command sets the context for the execution of the appropriate sensory processing, world modeling, planning and value judgment, and behavior generation processes [Albus 2002] in order to decide on the output command to subordinate(s). All of these processing elements are present in each agent control module. This generic structure facilitates comprehension of any control system built from it. The developer or the domain expert can locate any piece of knowledge in the structure and see how it is processed.

Behavior Generation is typically done via Finite State Machines (FSM's) or rules governing costs for nodes and edges in graph-search based planning techniques. These approaches cluster and order the knowledge rules specific to a particular task goal for an agent control module. Part of the implementation procedure is to determine which rules apply to each particular subtask activity at each level in the hierarchy. This is a natural outcome of the task decomposition process.

The World Model and Knowledge Base**.** The processing in an agent control module mimics the model of the human conscious/subconscious mind described above, i.e. it performs sensory processing (SP) on input data to recognize and classify objects and place them into a world model. The world model (WM) function processes the world state in the context of the present command to cause the behavior generation (BG) function to output a new command to a subordinate agent control module, or to request the planner to simulate the possible future world states and cost to achieve them, do a value judgment (VJ) based on cost analysis and suggest a reasonable plan to the BG function.

Analysis of the FSMs' condition value judgments is part of the RCS technique to define the necessary knowledge base and primitive world model elements. Using the on-road driving example, the rule that states if the "ConditionsAreGoodToPass" then the BG function can command the subordinate to "MoveToLeftLane", has the value judgment "ConditionsAreGoodToPass." To perform this value judgment, all of the rules-of-the-road concerning passing are consulted; weather and road surface conditions are sensed to determine appropriateness of this maneuver; oncoming vehicles are recognized out to considerable distances, etc. All of the relevant knowledge and specific reasoning in the context of this activity is gathered from the expert. From this, all of the primitive world model elements that need to be measured (such as vehicles, speed, direction, location, signs) are determined. These primitive world model elements then set the requirements for the SP system needed to support these control tasks.

4 Representation and Methodology Comparisons

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 [Arkin 1989], with most architectures lying somewhere in between these 2 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 [Balakirsky 2003]. This approach

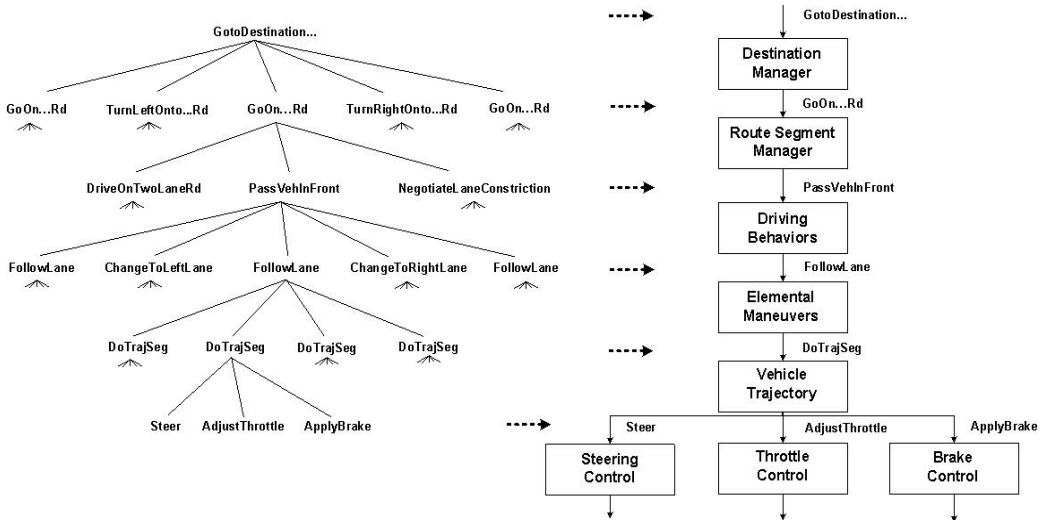


Figure 1:RCS implementation creates a hierarchy of agent control modules (right side) that are the execution engine for the task decomposition (left side)

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. [Brooks 1986]

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 then their reactive cousins [Mataric 1997] 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 [Lacaze et.al. 1998], 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 system was implemented using the RCS architecture, various other deliberative architectures such as CIRCA [Musliner et.al. 1993] and the Three-level Intelligent Machine [Saridis 1988] 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 [Swartout and Gill 1996], CommonKADS [Schreiber et.al. 1994] 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 KA process with classical control system design.

5 Application Example

In this section, we elaborate on the example pertaining to passing another vehicle on a two-lane undivided road.

Develop scenario with a domain expert. It is beneficial to drive in a vehicle with the domain expert and to have them speak through the process of determining when it was appropriate to pass. These specific conditions that spawn behaviors often change slightly depending on the personality of the driver, but we try to generalize the behavior to its fundamental components when encoding it in the control system.

Develop the task decomposition hierarchy. Before we can encode the knowledge needed to pass on a two-lane undivided road, we must understand and build an initial overall task decomposition hierarchy for on-road driving. The task decomposition hierarchy is often structured around a hierarchy of goals within the overall task at different levels of abstraction. An example of the on-road driving task decomposition hierarchy is seen in Figure 1, left side. The passing scenario is one of many scenarios that is used to develop this task decomposition hierarchy.

Determine the conditions that cause you to perform an action and the sub-actions that are needed to perform that action. In the case of passing, the actions that need to be performed are fairly straightforward; namely, change to left lane, follow left lane for some period of time, and change to right lane. This is shown in Figure 1. However, the conditions of when to start this sequence of actions and when to progress from one action to the next is much more difficult to understand.

In speaking with domain experts, there are two general conditions that must be true to initiate a passing operation: our vehicle desires to pass and conditions are good to pass. The next logical question to the domain expert is "When are the conditions good to pass?" Five sub-conditions must be true: 1) it is legal to pass, 2) the environmental weather and visibility conditions are conducive to passing, 3) the situation in front of our

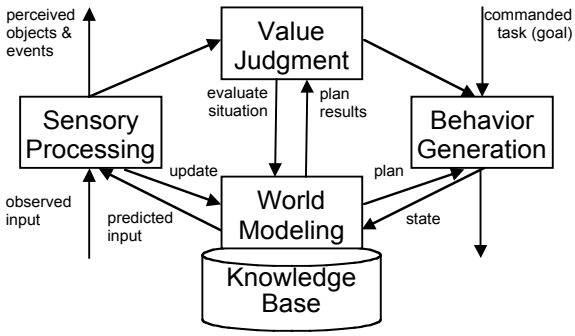


Figure 2: Generic Agent Control Module

vehicle is OK to pass (other vehicles, pedestrians, and objects do not hinder us), 4) the situation in back of our vehicle is OK to pass (the vehicle behind us is not passing or tailgating), and 5) oncoming traffic allows us to pass safely (we have time to get around the vehicle in front of us). Each of these sub-conditions is broken further down into sub-conditions until the point where we have identified the objects in the environment, and their pertinent attributes, that are relevant to this passing action.

Define the contents and structure of knowledge base. The objects and attributes discovered in the previous step set the requirements for the knowledge base that underlies the system. Following through with the scenario of passing on a two-lane undivided road, in order to evaluate the conditions mentioned in the previous step, the knowledge base must contain concepts such as other vehicles, including their speed, direction, and intention, pedestrians, lane markings and type, weather conditions, and signs. Once these concepts are captured in the knowledge base, they can be also used to derive requirements for sensor processing.

Develop a control hierarchy. The control hierarchy is the execution engine for the task decomposition. This hierarchy is built bottom-up (unlike the task decomposition hierarchy which is built top-down). The control hierarchy is based upon the actuators that must be grouped and coordinated to accomplish a given task. The actuators in the case of on-road driving include steering, braking, throttle, gear shift, turn signal, etc. The control hierarchy is shown in Figure 1 (right side) along with the mappings between the task decomposition hierarchy and the control hierarchy.

Test. Fundamental to the success of any system is testing. During the testing procedure, we determine if the vehicle was able to recognize the conditions in the world properly to trip appropriate behaviors, if the behaviors executed in a suitable way, and if we encountered situations that were not explicitly accounted for in the system.

6 Software Engineering Tools

Software engineering tools can facilitate systematic and efficient approaches for implementing RCS based systems. Over the years, there have been several thrusts in applying different software engineering tool paradigms to facilitate RCS applications. A Generic Shell [Huang et.al. 2000] approach that develops code frames for the RCS generic agent control module is used. C++ and Unified Modeling Language (UML) [Huang et.al. 2001b] versions of the generic shell have been implemented. Additional commercial tools, such as Control Shell, have also been used to model RCS control modules [Horst 2000]. Advanced Technology and Research Corporation has developed RCS tools that heavily focus on the Behavior Generation aspects of RCS and feature open, detailed task and world knowledge and visualization of real-time execution [Balakirsky and Messina 2002]. Formal methods have been researched, including the Rapide Architectural Description Language [Messina et.al. 2000] and the precise specification of RCS components in order to ensure conformance to the RCS reference architecture and to facilitate composition of systems. [Horst 1997; Messina et.al. 1999]

7 Summary

The RCS software engineering methodology is heavily based on understanding of human information management. The primary goal is to access, represent, and implement task knowledge for real-time control systems in a manner that matches the human mind's own mechanisms for processing information. The RCS attributes contribute to make the system accessible to modifications not only by the original system designer but by others, as a result of the generic agent control module and the hierarchical task decomposition architecture.

Product/Company Disclaimer

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

References

J. Albus, The NIST Real-Time Control System (RCS): An Application Survey. In *Proceedings of the 1995 AAAI Spring Symposium Series*, 1995.

J. Albus, 4D/RCS A Reference Model Architecture for Intelligent Unmanned Ground Vehicles. In *Proceedings of SPIE Aerosense Conference*, 2002.

J. Albus and A. Meystel, Engineering of Mind. John Wiley & Sons, Inc., 2001.

R. C. Arkin, Towards the Unification of Navigational Planning and Reactive Control. In *AAAI 1989 Spring Symposium on Robot Navigation*, 1989.

R. C. Atkinson and R. M. Shiffrin, Human Memory: A Proposed System and Its Control Processes. In K.W. Spence and J.T. Spence, *The Psychology of Learning and Motivation, Vol 2,* Academic Press 1968.

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

S. Balakirsky and E. Messina, A Simulation Framework for Evaluating Mobile Robots. In *Proceedings of the 2002 Performance Metrics For Intelligent Systems (PerMIS) Workshop*, pgs. 71-78, 2002.

R. Brooks, Achieving Artificial Intelligence Through Building Robots. Massachusetts Institute of Technology, 1986.

K. Dunbar and C. D. Schunn, The Temporal Nature of Scientific Discovery: The Roles of the Priming and

Analogy. In *Proceedings of the 12th Annual Conference of the Cognitive Science Society*, pgs. 93-100, 1990.

J. Horst, Precise Definition of Software Component Specifications. In *Proceedings of the 7th Symposium on Computer-Aided Control System Design*, 1997.

J. Horst, Architecture, Design Methodology, and Component-Based Tools for a Real-Time Inspection System. In *Proceedings of the 3rd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2000)*, 2000.

H. Huang, J. Albus, W. Shackleford, H. Scott, T. Kramer, E. Messina, F. Proctor, An Architecting Tool for Large-Scale System Control With an Application to a Manufacturing Workstation. In *proceedings of the 4th International Software Architecture Workshop, in conjunction with the International Conference on Software Engineering 2000 (ICSE 2000)*, 2000.

H. Huang, E. Messina, H. Scott, J. Albus, F. Proctor, W. Shackleford, Intelligent System Control: A Unified Approach and Applications. In *C.T. Leondes, The Technology of Knowledge Management for the 21st Century*, 2001a.

H. Huang, E. Messina, H. Scott, J. Albus, F. Proctor, W. Shackleford, Open System Architecture for Real-Time Control Using a UML Based Approach. In *Proceedings of the 1st ICSE Workshop on Describing Software Architectures with UML*, 2001b.

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.

A. Lacaze, K. Murphy, M. DelGiorno, Autonomous Mobility for the DEMO III Experimental Unmanned Vehicle. In *Proceedings of the AUVSI*, 2002.

M. J. Mataric. Behavior-Based Control: Examples From Navigation, Learning, and Group Behavior. Journal of Experimental and Theoretical Artificial Intelligence, 9(2-3): 323-336, 1997.

E. Messina, C. Dabrowski, H. Huang, J. Horst, Representation of the RCS Reference Model Architecture Using an Architectural Description Language. In *EUROCAST '99, 798 (Lecture Notes in Computer Science)*, 2000.

E. Messina, J. Horst, T. Kramer, H. M. Huang, T. Tsai, E. Amatucci, A Knowledge-Based Inspection Workstation. In *Proceedings of the 1999 IEEE International Conference on Information, Intelligence, and Systems*, 1999.

G. A. Miller. The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information. The Psychological Review, 6381-97, 1956.

D. J. Musliner, E. H. Durfee, K. G. Shin. CIRCA: A Cooperative Intelligent Real-Time Control Architecture. IEEE Transactions on Systems Man and Cybernetics, 23(6): 1561-1574, 1993.

G. N. Saridis, On the Theory of Intelligent Machines: A Survey. In *Proceedings of the 27th IEEE Conference on Decision and control*, pgs. 1799-1804, 1988.

A. T. Schreiber, B. J. Wielinga, R. De Hood, J. M. Akkermans, W. Van de Velde. CommonKADS: A Comprehensive Methodology for KBS Development. IEEE Expert, 9(6): 28-37, 1994.

B. Swartout and Y. Gill, Flexible Knowledge Acquisition Through Explicit Representation of Knowledge Roles. In *1996 AAAI Spring Symposium on Acquisition, Learning, 965 and Demonstration: Automating Tasks for Users*, 1996.

E. Tulving, Episodic and Semantic Memory. In *E. Tulving and W. Donaldson, Organization of Memory*, 1972.

E. Tulving. Precis of Elements of Episodic Memory. Behavior and Brain Sciences, 7(2): 223-268, 1984.

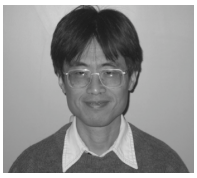
Contact

Tony Barbera
Intelligent Systems Division
National Institute of Standards and Technology
Gaitherburg, MD 20899
tony.barbera@nist.gov

Authors

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

Elena Messina is Knowledge Systems Group Leader In the Intelligent Systems Division at NIST. She has over 20 years experience in robotics and in software engineering for complex systems.



Hui-Min Huang has been an engineer at NIST for 15 years and conducts research on software architectures and UML based, scenario driven methods for intelligent systems such as unmanned vehicles. He coordinates/participates in architectural standards working groups for interoperability, performance, and autonomy levels and has over 50 publications.

Craig 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.

Stephen Balakirsky received the Ph.D. degree from the University of Bremen, Germany in 2003. He is currently a researcher in the Intelligent Systems Division of the National Institute of Standards and Technology. His research interests include planning systems, knowledge representations, world modeling, and architectures for autonomous systems.