Proceedings of the 7th International Workshop on Computer Aided Systems Theory and Technology 1999 (EUROCAST'99), Vienna, Austria, Sept. 29 - Oct. 2, 1999.

# REPRESENTATION OF THE RCS REFERENCE MODEL ARCHITECTURE USING AN ARCHITECTURAL DESCRIPTION LANGUAGE

**E. Messina, C. Dabrowski, H. Huang, J. Horst**

National Institute of Standards and Technology
100 Bureau Drive, Stop 8230
Gaithersburg, MD USA 20899-8230
Tel. +1-301-975-3510, FAX +1-301-990-9688
E-mail: elena.messina@nist.gov

## 1. INTRODUCTION

The Real-Time Control System (RCS) provides a reference model architecture for the construction of intelligent systems. RCS specifies construction of complex systems based on computational nodes that contain the same basic processing elements. Researchers at the National Institute of Standards and Technology (NIST) have been investigating better means of documenting and communicating the RCS architecture. NIST is interested in tools that help verify conformance to RCS and promote reuse within the RCS community. An investigation was conducted into the use of Architectural Description Languages (ADLs) as a means of providing a more rigorous description of RCS and potentially addressing the communication, conformance validation, and reuse requirements.

## 2. THE REAL-TIME CONTROL SYSTEM REFERENCE MODEL ARCHITECTURE

The Real-Time Control System reference model architecture has been used to develop a variety of complex systems, particularly in manufacturing automation (Albus, 1995). Recently, a variant, 4-D/RCS (Albus, 1997) has been selected as the software architecture for the U. S. Department of Defense Demo III Unmanned Vehicle Program (Shoemaker and Bornstein, 1998). This version of RCS was chosen to be represented by an ADL.

RCS prescribes a building-block approach to designing and implementing systems. The building blocks are control nodes that contain the same basic elements in order to accomplish complex behaviors. These elements, shown in Figure 1, are behavior generation (BG), world modeling (WM), value judgement (VJ), and sensory processing (SP). Each node receives goals from its superior and, through the orchestration of BG, WM, VJ, and SP, generates a finer-resolution set of goals for its subordinate nodes. The RCS control node uses an estimated state of the world, generated via SP and WM, to assess its progress with respect to the goals it was given and to make necessary adjustments to its behavior. BG's sub-modules are the job assigner (JA), a set of plan schedulers (SC), a plan selector (PS), and a set of executors (EX). One SC and EX exist for each subordinate controlled by a particular RCS node. JA decomposes incoming commands into job assignments for each of its subordinates. Each SC computes a schedule for its given assignment. JA and SC produce tentative plans based on available resources. PS selects from the candidate plans by using WM to simulate the execution of the plans and VJ to evaluate the outcomes of the tentative plans. The corresponding EX executes the selected plan, coordinating actions

among subordinates and correcting for errors between the plan and the state of the world estimated by the WM.

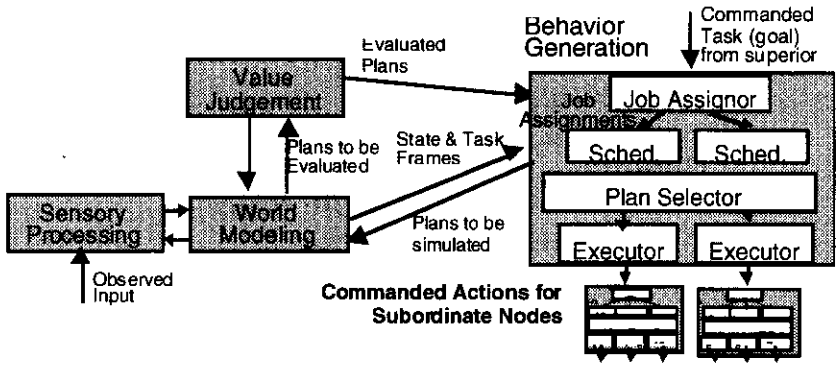


**Figure 1 - An RCS Control Node**

## 3. ARCHITECTURAL DESCRIPTION LANGUAGES

### 3.1 General Concepts

A software architecture may be considered an abstract specification of a system's functional components, their behavior and external interfaces, and interconnections. ADLs typically provide a defined language and syntax with which to specify the following main categories of architectural elements. 1.) Software components: definition of the interface messages and commands accepted and sent, constraints on the interface, and behavior of the component in response to events or messages; 2.) Connections between components: an architecture may, at minimum, be defined by connection of messages emitted by one component and received by another. Constraints on message content and sequences may also be specified; 3.) System Behavior: ADLs may provide support for specifying the behavior of component interfaces, connections, and internals; and 4.) Architectural Styles: ADLs may provide support for constraining how components can be connected and what system topologies may be described. E.g., in RCS each node has a single supervisor. Some ADLs also provide mechanisms for formal verification of internal consistency and completeness of the architecture specification. Certain ADLs permit execution of the architecture specification, enabling designers to evaluate the sequencing and behavior of their system design. ADLs may provide tools for verifying whether the design for an implementation architecture conforms to a reference architecture or not.

### 3.2 *Rapide*

*Rapide* (Luckham, 1996) is an ADL and tool set developed at Stanford University. We selected this ADL due to its capabilities for representing and simulating real-time system designs and for supporting most of the features described in section 3.1. Within *Rapide*, a system architecture is defined by *connections* between *interfaces* of components, whose internals are defined by *module* specifications. Component interface types can be defined, including events generated and received by the components and a description of the component's behavior. *Constraints* can be defined for an interface, such as dependencies between events or limitations on parameter values. The internal *behavior* of a module

defines how it responds to events received by the interface, generates events sent by the interface, and obeys constraints defined in the interface. *Rapide* provides language support for defining event types, causality, and behavioral constraints. Features exist for representing parallelism and non-deterministic behavior. The defined architecture can be executed in simulation. Partially Ordered Set of Events (POSETs) are aggregations of the event dependencies generated through simulation and can be used to evaluate the execution and compare a concrete architecture to an abstract reference model.

## 4. SPECIFICATION OF AN RCS CONTROL NODE IN *RAPIDE*

### 4.1 Abridged Example Specification

For our study, we prepared a specification of an RCS control node and its constituent components BG (JA, SC, EX), WM, VJ, and SP. For each component, an interface and module specification were created (Dabrowski et al., 1999). Connections between the components were defined, along with some behavior specifications that enabled simulation of the architecture. The entire specification was over 1000 lines of *Rapide* code. A brief excerpt illustrating the Job_Assignor Interface is shown in Figure 2. Incoming and outgoing events are defined via the ACTION IN and OUT statements. JA accepts a task (Do_Task) containing a command frame defining what is to be accomplished. Through Fetch_Task_Frame, JA receives a Task_Frame containing the information necessary to perform the task assigned. JA outputs a command to its subordinate SCs to create schedules. The behavior definition indicates the event causality. $Event_1$ |> $Event_2$ is the syntax in *Rapide* for indicating that $Event_1$ causes $Event_2$ to occur. The receipt of a Do_Task command triggers a request for the task information (Fetch_Task_Frame). ?Task is a variable placeholder for the task and ?TF is the variable for the Task_Frame. When a Task_Frame is received (RCV_Task_Frame), a command to decompose the task frame into commands for the various subordinates is triggered. An example of a constraint on the behavior is shown. The || notation indicates causally independent events. The constraint says that Do_Task and Schedule_Job are not allowed to be independent of each other.

```
TYPE Job_Assignor_Interface IS INTERFACE;
    ACTION
        IN
              Do_Task (Task : Task_Command_Frame),
              RCV_Task_Frame (Task : Task_Command_Frame; TF : Task_Frame),
        OUT
              Schedule_Job (Job : Task_Command_Frame),
              Fetch_Task_Frame (Task : Task_Command_Frame),
              Decompose_Task_Frame (TF : Task_Frame),
    BEHAVIOR
        (?Task : Task_Command_Frame) Do_Task (?Task) |>   Fetch_task_frame (?Task);
        (?Task : Task_Command_Frame; ?TF : Task_Frame)
             RCV_task_frame (?Task, ?TF) |> Decompose_task_frame (?TF);;
    CONSTRAINT
        NEVER (?Task : String; ?Job: String) Do_Task (?Task) || Schedule_Job(?Job);
END;
```

**Figure 2: Excerpt from the *Rapide* specification of Job Assignor**

### 4.2 Results of Experiment

A specification of the RCS control node was successfully constructed using *Rapide*. All of the main aspects of the RCS reference architecture were representable and executable in

simulation. To date, this is the most rigorous representation of the RCS reference model architecture. The specification was presented to a group of RCS domain experts, unfamiliar with ADLs. After overcoming their unfamiliarity with the notation, they were able to understand the specifications, aided by the simulation capabilities in *Rapide*. Their assessment of the specification found that it captured RCS correctly. The process of generating the specification itself was beneficial. The domain expert working on the study extended the RCS reference model to include behaviors and interfaces that were created for the *Rapide* specification. For example, the Fetch_Task_Frame operation preceding task decomposition in JA was not explicitly part of the RCS reference model, but was found to be significant enough to be added to it. The experiment included a verification that a system design conformed to the RCS reference model. In *Rapide*, this is accomplished via mapping of events from the concrete design to the abstract architecture. The concrete and abstract architectures are executed in simulation. Their resulting POSETs are compared and inconsistencies are flagged. This approach found a violation of the constraint on causal dependence between Do_Task and Schedule_Job in a concrete architecture.

## 5. FUTURE DIRECTIONS

We successfully built a specification of the RCS reference model architecture in the *Rapide* ADL. This effort yielded the most formal definition of RCS thus far and provided advantages such as simulation capabilities, some conformance validation, and insights into extensions to the reference model. Research areas remain that could yield benefits to the specification of complex architectures such as RCS. These include being able to specify structure, better means of communicating system behavior, stronger support for real-time computation, more rigorous checks on conformance, and syntax extensions to make specifications more accessible to application domain experts. Finally, addition of defined language features for enabling systematic reuse of architecture specifications would promote the use of ADLs in software engineering design environments.

## 6. REFERENCES


Albus, J. S. (1995) The NIST Real-time Control System (RCS) An Applications Survey. In: Proc. of the AAAI 1995 Spring Symposium Series, Stanford Univ., Palo Alto, CA.

Albus, J. S. (1997) 4-D/RCS: A Reference Model Architecture Demo III. National Institute of Standards and Technology, Gaithersburg, MD, NISTIR 5994.

Dabrowski, C., Messina, E., Huang, H., Horst, J., (1999) Using Architectural Description Languages to Formalize the NIST 4-D/RCS Reference Model Architecture. National Institute of Standards and Technology, Gaithersburg, MD, NISTIR to appear.

Luckham, D. (1996) *Rapide*: A Language and Toolset for Simulation of Distributed Systems by Partial Ordering of Events. Stanford Univ. Palo Alto, CA. CSL-TR-96-705.

Shoemaker, C., Bornstein, J. (1998). Overview of the Demo III UGV program. In: Proc. of the SPIE Robotic and Semi-Robotic Ground Vehicle Technology, Vol. 3366.


DISCLAIMER – Certain products are identified in this report to describe our study adequately. Such identification does not imply recommendation or endorsement by NIST.