188578 S16-63 68

# An Architecture for Heuristic Control of Real-Time Processes

P. Raulefs and P.W. Thorndyke
FMC Corporation
Santa Clara, CA 95052

FY 563409

**Abstract** *Process management* combines complementary approaches of heuristic reasoning and analytical process control. Management of a continuous process requires monitoring the environment and the controlled system, assessing the ongoing situation, developing and revising planned actions, and controlling the execution of the actions. For knowledge-intensive domains, process management entails the potentially time-stressed cooperation among a variety of expert systems. By redesigning a blackboard control architecture in an object-oriented framework, we obtain an approach to process management that considerably extends blackboard control mechanisms and overcomes limitations of blackboard systems.

## 1. Introduction

Many future military and space applications will require control of autonomous or semiautonomous systems operating in dynamic environments. *Real-time heuristic control* is the task of applying knowledge-intensive reasoning methods to supervise and manage such dynamic systems. Effective control requires monitoring and assessing rapidly changing situational data from the environment in which the system operates, developing and evaluating planned actions, and executing those actions to achieve desired goals. This requires the control system to maintain a model of the operating environment and to interpret sensor data and planned actions in light of this model.

For operating environments too complex to model using traditional process control models, knowledge-based qualitative models can provide an effective means of approaching the tasks of dynamic situation assessment and planning. However, existing AI-based techniques for data interpretation, situation assessment, and planning cannot accommodate requirements for operating in time-stressed situations where critical conditions and assumptions may be varying dynamically. Further, existing methods cannot easily accommodate the synchronization of interacting processes engaged in situation assessment, planning and execution control.

Blackboard control systems [4, 2] provide a first step towards solving these problems. Our analysis (Section 3.2) shows, however, that they lack the facilities needed to meet requirements of real-time responsiveness and hybrid, layered system architectures.

After analyzing the generic real-time heuristic control problem more closely in Section 2, we present in Section 3 the Heuristic Control Virtual Machine, or HCVM, as an approach that considerably extends blackboard control architectures to solve problems associated with their shortcomings. The underlying idea of the HCVM is to cast a blackboard control architecture into an object-oriented framework [3] and then exploit the additional features provided by object-oriented programming to design solutions to the above problems.

We show how flexible, even dynamically variable interaction modes between objects can be used to achieve real-time responsiveness. Using objects with standard interfaces as 'wrappers' around both heuristic and analytical procedures allows building hybrid systems that integrate knowledge-based and conventional process control components. Control reasoning using explicit notions of time and temporal relations about activities in *both* the controlled environment *and* the control system provides an approach to meet timeliness requirements.

Finally, current experience suggests a repertoire of methods applicable to a broad domain of applications. Section 4 discusses how the HCVM could provide a computational model for a more comprehensive software engineering environment for building process management applications.

## 2. The Process Management Problem

Process control is concerned with monitoring and assessing changes in dynamic systems, followed by planning and executing actions to control dynamic systems. Analytical techniques employed by conventional process control utilize mathematical models of controlled systems, expressed in terms of differential equations of time-varying functions relating sensor data and controlled variables. Conventional process control is limited to applications where such models and sufficient sensor data are available. Dynamic systems with this property are often successfully managed by human operators using heuristic expertise instead of analytical reasoning. In fact, a great number of process control systems operate in environments where they are complemented by human operators. This observation suggests to view analytical and heuristic process control as complementary. We refer to *process management* as the combination of both.

Heuristic techniques play a dual role in process management: *Supervisory control* applies heuristic judgment to make decisions about results elaborated by analytical *or* heuristic methods, and *direct control* interprets sensor data to make and execute control decisions.

Applications impose two types of requirements on process management systems. *Functional requirements* consist of functions and tasks to be performed. *Performance requirements* constrain time and other resources available to carry out tasks. *Real-time performance* is a particular challenge difficult to overcome by knowledge systems.

Several characteristics of intended application domains influence functional and performance requirements. We consider domains where up to tens of thousands of sensors generate data about many fewer subsystems or components. Data sources are distributed over space and time, such as satellites producing bursts of data every few hours. Both continuous and discrete data occur with often dynamically varying data rates and urgency. Data are unreliable because of potential sensor and communication failures and noise.

Functional requirements concern monitoring the environment, situation assessment, planning and re-planning, plan execution control, and coordinating and managing the system's own activities.

1. **Monitoring.** The monitoring function includes:
   * *Monitoring the environment:* Receive incoming data; filter, fuse, interpret and abstract incoming data for data reduction, critical event and alert generation to interrupt, override or pre-empt current activity.

   * *Monitoring system performance:* Inspect current process management activities, possibly in conjunction with results of monitoring incoming data, to detect critical control events and raise control alerts.

2. **Situation Assessment.** The process management system must diagnose the current situation of the environment; determine how it evolved from previous history, and predict future developments; determine goals to be achieved by process management activity. Situation assessment is done dynamically, i.e. it extends, modifies and revises previous assessment, and it will have to incorporate new information when it becomes available.

3. **Action Planning.** From results of Monitoring and Situation Assessment, produce a plan of process management actions to reach the goals. Plans will include *conditional actions* with alternatives if conditions do not hold. *Re-planning* modifies plans to adapt to dynamically changed situations in the environment (known when new data have been received) or the process management system (when new assessments, critical events have been produced).

4. **Plan execution and Monitoring.** Plans produced by Action Planning will be high-level descriptions yet to be compiled into executable instructions to effectors or the user interface. Monitoring plan execution determines whether actions are carried out as specified and have anticipated effects.

5. **Managing Own Activities.** The system must allocate resources to the above tasks, start their activities, and monitor their progress. To do so in a timely fashion, it must assign priorities, schedule activities, and manage interrupts.

The performance requirements specific to real-time systems concern their responsiveness to changes in their environment:

1. **Timeliness.** All scheduled activity must be completed sufficiently early to have its desired effect on the environment. Any task that completes successfully, but too late to have its intended impact violates the timeliness requirement.

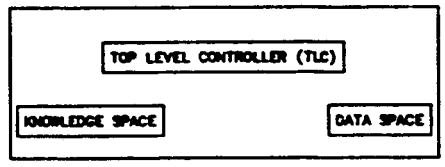
2. **Interruptibility.** Interrupts suspend or terminate activities in favor of activating others. Interrupts require a model separating atomic, non-interruptible from composite, interruptible tasks that is compatible with the timeliness requirement. Composite tasks can be interrupted to give control to tasks of higher priority. Interrupts are accompanied by strategies for orderly resumption of tasks that preserve the integrity of the overall task.

Real-time performance cannot be achieved by merely optimizing hardware and software to operate fast. For timely responses, a process management system has to plan and monitor the temporal performance of its own activities together with those of its environment. Interruptibility requires the system architecture to provide facilities for setting up tasks of variable and even dynamically varying size, and for managing suspension and resumption of tasks to transfer control. Interruptibility contributes to achieve timeliness by providing a mechanism to postpone tasks and move up others to complete more time-stressed tasks earlier.

# 3. HCVM: An Architecture for Real-Time Process Management

## 3.1 HCVM Overview

The Heuristic Control Virtual Machine (HCVM) is a knowledge-processing operating system designed as an environment for engineering real-time, knowledge-intensive process management applications. The HCVM (see Fig. 3-1) consists of three collections of objects: The top level controller (TLC), the knowledge space and the data space. The knowledge space consists of knowledge handlers that embody heuristic expertise and analytical procedures. The elements of the data space are data handlers carrying information obtained from outside communication or produced by knowledge handlers. All activity of knowledge and data handlers is managed by the TLC.




HEURISTIC CONTROL VIRTUAL MACHINE

```
<TOP LEVEL CONTROLLER>  :?:= <knowledge mgr> <data mgr> <control mgr>
                             <control plan>


<KNOWLEDGE SPACE>    ::= "a collection of knowledge handlers"
<KNOWLEDGE HANDLER>  ::= <trigger condition> <precondition> <body>
<TASK HANDLER>       ::=                     <precondition> <body>

<DATA SPACE>         ::= "a collection of data handlers"
<DATA HANDLER>       ::= <information> <local data manipulation methods>
```

**Figure 3-1:** HCVM Overview

A Knowledge handler is activated by the TLC when (1) it has been triggered, (2) it has been scheduled for execution by the TLC, and (3) is in a state where its <precondition> is true. Trigger conditions are propositions about information stored in data handlers. Executing a knowledge handler consists of carrying out its <body> that may result in changes in the data space and communication with the environment external to the HCVM. Information stored in data handlers changes when knowledge handlers produce such changes (e.g., after inferring a new diagnosis), or when new information from the external environment is communicated to the HCVM and affects a particular data handler (e.g., after having received new sensor data). When a data handler has received some update information, it may execute some internal code to, for example, check consistency, perform some smoothing and averaging operation on new and previous values, or determine qualitative abstractions from numerical values.

The TLC repeatedly executes a cycle of invoking the knowledge, data and control manager modules. The knowledge manager arranges executing knowledge handlers that have been marked executable by the control manager. Then the data manager carries out resulting changes in the data space. The control manager maintains a control plan that specifies which knowledge handler(s) are to be executed next and in which order. To arrive at a control plan, the control manager may perform extensive control planning about tasks to be done in the external environment and the HCVM. To do this, the control manager may employ knowledge and data handlers, thus restricting a series of control cycles to control reasoning.

### 3.2 Shortcomings of Blackboard Architectures

The HCVM as described above is essentially a *blackboard control architecture* [4] cast into an object-oriented programming *framework*. Knowledge handlers correspond to knowledge sources, the data space to a combined domain and control blackboard, and the TLC includes the facilities for blackboard control reasoning. However, this does not suffice to meet the requirements described in the previous section. We now explain the mechanisms by which the HCVM considerably extends blackboard control architectures in connection with the problems they are meant to solve.

Considering the requirements posed in section 2, blackboard architectures have particular difficulties in handling the following problems:

**Interruptibility** of tasks in any computational model is determined by the grain size of non-interruptible, atomic units. Knowledge sources are such units in blackboard systems: Once a knowledge source is activated, it is executed until it terminates. A knowledge sources provides a facility to collect closely related expertise and procedures so that interactions between knowledge sources are restricted to less closely related capabilities. This principle of organizing knowledge can be in conflict with timeliness and interruptibility requirements to structure activity into tasks of sufficiently small grain size.

**Timeliness** is difficult to achieve because blackboard control is excessively expensive when scheduling tasks does not require extensive control reasoning, such as for tasks that are known to be executed in a sequence, one after another.

**Responsiveness** to an incoming glut of data with dynamically varying information density and urgency is difficult to achieve with blackboard systems. Incoming data need to be analyzed first before control planning can schedule activities to respond, implying at least two execution cycles of the blackboard system that even may be interrupted by high-priority alerts.

**Hybrid and layered system architectures** where activities such as blackboard control, or those of individual knowledge sources are handled by conventional "control boxes" or blackboard systems in themselves are difficult to build as blackboard Systems. Interactions, including property inheritance, switching activations from lower to higher level tasks and vice versa, and managing interrupts between different levels have to go through blackboard control cycles although this is often not an adequate mechanism to use.

To help solve these problems, the HCVM introduces several additional mechanisms and features.

### 3.3 Synchronous and Asynchronous Interaction Modes between Handlers

Besides the *blackboard mode* of triggering, scheduling and then invoking knowledge handlers, the HCVM provides for two additional modes of interaction between modules: First, *direct message passing* activates a module by sending it a message, as usual for object-oriented systems. The second is *remote routine call* by which an object has another object executes a routine and then return a result to the caller. For the first two modes, activities of objects causing those of others are not directly linked to the activated objects. In the blackboard mode, both activities are separated by arbitrary other activities. For the blackboard and the message passing mode, invoking and invoked object may logically and physically proceed concurrently, and their activities are not synchronized. For remote routine call, the caller and callee are synchronized by the caller being suspended until the the callee returns control to resume the caller.

The HCVM adds an additional feature to these standard mechanisms of interaction between modules: interaction modes may vary dynamically. For example, a knowledge handler may be invoked by blackboard control triggering and activating it, by some other handler sending it a message (knowledge handlers are *named* objects), or by some other handler invoking a routine (e.g., to execute a set of rules) that the knowledge handler provides.

Handlers in the knowledge space that are not supposed to be invoked by blackboard control are realized as **task handlers** in the HCVM: A task handler is like a knowledge handler, but it lacks a <trigger condition> (see Fig. 3-1).

Knowledge and task handlers may have bodies built up from task handlers, with control passing from one task handler to another in a way that may vary dynamically (e.g., by evaluating conditions to decide which one is the next to execute).

Although these facilities appear to be straightforward extensions of blackboard systems by incorporating additional standard module interaction modes, they open a significant problem of designing coherent interrupt management schemes. For example, a knowledge handler may be interrupted after having finished some task handler. Resuming execution of this knowledge handler may occur after significant changes, suggesting to not simply continue with the next previously scheduled task handler. A thorough discussion of techniques for such capabilities goes beyond the scope of this paper. We note, however, that they provide flexible interrupt schemes to attend rapidly changing needs in an optimal way.

### 3.4 Multiple HCVMs and Heterogeneous Handlers

Large-scale process management applications may lead to knowledge handlers of considerable size, blackboard control that is extremely complicated, and I/O-communication that is difficult to handle within a single HCVM-object because of immense data rates. Each of these activities is best performed by a system providing all the facilities offered by an HCVM.

The other problem of process management applications is that knowledge-based reasoning closely interacts with traditional process control equipment, such as PID-controllers and other types of "control boxes".

The object-oriented nature of the HCVM allows quite easily to treat these issues by two approaches:

* As an HCVM is just an object, knowledge and task handlers, as well as a TLC or its components may in themselves be full-fledged HCVMs. As mentioned above, a problem beyond the scope of this paper is how to manage interactions (such as property inheritance and interrupts) between "ordinary" and HCVM-knowledge/task handlers.

* A knowledge or task handler may encapsulate a physical process control unit, where HCVM-interactions with such a handler may consist in communicating sensor data and set points.

### 3.5 Communication with External Environment

Unforeseen dynamic changes in an external environment with the properties described in section 2 lead to two problems of dynamically changing needs for communication between environment and process management system.

* Varying information density of incoming data require process management to allocate varying resources for buffering and preprocessing. Much of preprocessing concerns the monitor functions of screening and compacting input data, where screening discards less relevant data as early as possible.

* Varying urgency of incoming data needs to be determined and anticipated by process management for planning and re-planning its tasks and resource allocation. In particular, communication capacity has to be dynamically adjusted to varying urgency so that high-bandwidth communication is only devoted to messages of highest importance.

The HCVM provides four mechanisms of increasing complexity to support solving these problems:

1. I/O-communication is handled by a "Communication Manager" executed as part of the TLC-loop. For each TLC-cycle, the Communication Manager empties all I/O-buffers by distributing incoming data into the HCVM, and sending data in output buffers away. This method is adequate when information densities and urgencies of communicated data vary approximately in the same way as the amount of time that is spent on processing the other TLC-tasks.

2. I/O-communication is done by a knowledge handler triggered when data with specified properties (e.g., sufficient volume and urgency) appear in I/O-buffers. This is adequate as long as there is no need to differentiate between processing rates in different I/O-buffers.

3. I/O-communication is done by several knowledge handlers specialized to process data for particular I/O-buffers.

4. I/O-communication is done by knowledge handlers activated both synchronously and asynchronously.

### 3.6 Control Reasoning Support

Process management is concerned with three types of actions. *Domain actions* are carried out in the environment and their effects are observed in the process management system. *Control actions* are executed by process management to influence the course of domain actions. *Self control actions* are those guiding the process management system to perform its own tasks. The task of 'managing its own activities' (see section 2) recursively repeats monitoring, situation assessment, planning and plan execution.

The HCVM is well-suited to support control reasoning. Knowledge handlers embodying explicit control knowledge are triggered when data handlers record relevant conditions and events for environment and process management. A collection of such knowledge and data handler classes to be instantiated with creating an HCVM constitutes a *control framework* that drives the control manager in the TLC to select and schedule process management activities. Knowledge handlers could either themselves be of different grain-size, or be composites of knowledge/task handlers focusing on particular dynamic situations. The control manager of the HCVM dynamically manages priorities and temporal dependencies among knowledge handlers scheduled for execution, providing a mechanism to respond to dynamically changing needs.

## 4. Mapping Architecture to Application Structures

### 4.1 Multiple Agents Reasoning About Hierarchical Component Structures

We have applied the HCVM to the development of a continuous process management problem of mineral refinement. This domain has the following characteristics:

* The environment consists of hundreds of components with thousands of sensors gathering data about the states of components.

* Components are organized in a functional hierarchy where interactions among functions coincide with control and data flow.

* Each major component requires independent process management within given requirements dynamically influenced by other major components.

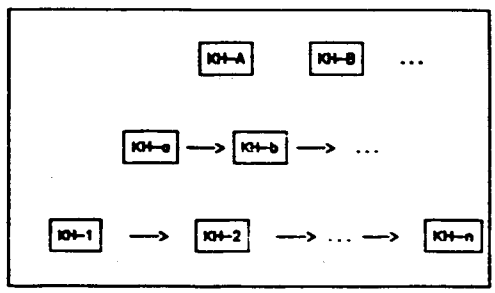



**Figure 4-1:** Hierarchy of Process Management Agents

Fig. 4-1 shows a process management system structure where a hierarchy of knowledge handlers reflects the component hierarchy. Each knowledge handler is an HCVM-object performing process management for a particular component. Higher-level knowledge handlers combine sensor data with monitoring and situation assessment results from lower-level agents to form more abstract, strategic assessments and plans.

Lower-level knowledge handlers (e.g., KH-1, KH-2, ... in Fig. 4-1) are executed in a fixed order which is implemented as sequential execution where trigger conditions are ignored. Interrupts, however, may lead to terminate execution of any such task, and resume execution of the *sequence* at any given point. Resumption is implemented as triggering knowledge handlers, scheduling the first in the sequence, and discarding the rest from the control plan (as subsequent elements of the sequence, they are executed, anyway).

### 4.2 Towards Process Management Engineering Environments

The HCVM provides a *computational model* in a more comprehensive architecture for real-time knowledge systems, as shown in Fig. 4-2. In this role, the HCVM is a software environment to implement the problem-solving frameworks that support building capabilities to perform the generic functions of process management described in Section 2.

**Example**

To describe how the HCVM supports such a system organization, we review how a real-time process management system implemented on an HCVM performs its tasks [6].

> Sensor data are received as separate streams of data packages in several input ports. A Communication Manager (see approach 1 in Section 2.5) turns data packages over to package-specific knowledge handlers for screening. Unless critical values are detected, packet handlers send relevant and possibly pre-processed data into the data space.
>
> Data handlers provide capabilities of a process data base management system: Data are stored in a way that supports retrieving historical information, such as "get the temperature distribution between the last sharp drop and subsequent sharp increase of pressure". Data handlers also perform data abstraction from numerical to qualitative values (quantization), smoothing and averaging operations, and locally decidable reliability checks.
>
> Knowledge/task handlers are activated upon the occurrence of changes in the data space, or upon the persistence of properties about data over a specified period of time. For situation assessment, knowledge/task handlers determine situations and events constituting higher-level descriptions about conditions and changes in the environment. Conditions and changes occur in the contexts of temporally extended patterns of situations and events, or *processes*. Situation assessment results in recognizing which processes are active, how they interact, and to which states they have progressed.
>
> Recognition of processes activates knowledge/task handlers that plan adequate responses. Ongoing planning may be interrupted and revised following alerts generated when critical values are observed when screening incoming data, or critical situations or events are inferred by situation assessment. Interrupts are frequently used to interleave situation assessment and planning. For example, planning may request more detailed information on the occurrence of an anticipated critical event before committing to on of several possible courses of action.

The example indicates how process management functions are accomplished by using generic capabilities, such as data abstraction, event and process recognition, response planning, and control management. A software engineering environment for process management would provide the capabilities comprising the problem-solving frameworks of Fig. 4-2.

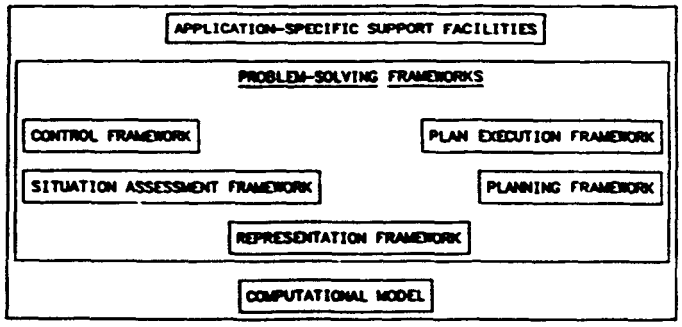



**Figure 4-2:** Organization of Real-time Knowledge Systems

## 5. Conclusions and Future Work

We have shown how the HCVM extends blackboard control architectures to overcome many of their shortcomings. Our work has so far indicated that the HCVM provides an adequate computational model to support the variety of tasks in process management, thus giving some evidence that it could be an appropriate basis for future real-time process management engineering environments. To become credible, this claim must be substantiated by actually building, applying and evaluating such systems.

As with blackboard architectures, the HCVM carries a significant computational overhead for simply managing its own structure. Unlike blackboard architectures, however, much of this overhead can be contained and even avoided by synchronous interaction modes between handlers. Empirical evidence needs to be gathered about the extent of such overhead, the improvements achievable by less expensive interaction modes, and the trade-offs with regard to lesser degrees of real-time responsiveness.

Another unexplored direction is distributing knowledge [5, 1] and data spaces across processors in distributed computing environments. A particularly interesting feature of the HCVM is that it involves highly synchronized communication via shared memory as well as loose message-passing interaction. Hardware architectures supporting both types of interaction provide appropriate support for these features, but there is a wide spectrum between tightly synchronous and loosely asynchronous interactions yet to be investigated.

## 6. Acknowledgements

The HCVM has been primarily developed by Teknowledge, Inc. in a cooperative project with FMC Corporation. The Teknowledge group was lead by Michael R. Fehling until October, 1986. The other members are Stephanie Forrest, Michael Wilber, and Wayne Caplinger. A previous version of the HCVM was developed by Michael Fehling and colleagues as the SCHEMER system. Michael Wilber (Teknowledge) implemented most of the HCVM, and Stephanie Forrest (Teknowledge) designed and implemented an application framework. Bruce D'Ambrosio, Anita Wong, and Khanh Ho from FMC contributed to the process management application system in our laboratory.

### References

[1] Durfee, E. H., Lesser, V. R., Corkill, D. D.
Increasing Coherence in a Distributed Problem Solving Network.
In *Proc. of the 9th IJCAI-85*, pages 1025-1030. August, 1985.

[2] Erman, L. D., Hayes-Roth, F., Lesser, V. R., Reddy, D. R.
The Hearsay-II speech understanding system: Integrating knowledge to resolve uncertainty.
*ACM Computing Surveys* 12:213-253, 1980.

[3] Goldberg, A. and D. Robson.
*Smalltalk-80: The language and its implementation.*
Addison-Wesley, Reading, Mass., 1983.

[4] Hayes-Roth, B.
A blackboard architecture for control.
*Artificial Intelligence* 26:251-321, 1985.

[5] Lesser, V. R. and Corkill, D.
Functionally accurate cooperative distributed systems.
*IEEE Trans. Systems, Man and Cybernetics* 1:81-96, 1981.

[6] Raulefs, P., D'Ambrosio, B., Fehling, M. R.,Forrest, S., Wilber, M.
Real-time process management of materials composition processes.
In *Proc. 3rd IEEE Conference on Applications of Artificial Intelligence.* IEEE, Orlando, Fla., Feb., 1987 (in press).