ANL/ED/CP--87433
CONF-951177--1

# Control and Supervision of a Complex Production Process Using Hybrid Systems Techniques

by

Humberto E. Garcia

Argonne National Laboratory-West
P.O. Box 2528
Idaho Falls, ID 83403-2528

ICECCS'95
First IEEE International Conference on
Engineering of Complex Computer Systems
Held jointly with 5th CSESAW, 3rd IEEE RTAW and 20th IFAC/IFIP WRTP
Ft. Lauderdale, Florida

November 6-10, 1995

*Work supported by the U.S. Department of Energy, Reactor Systems, Development and Technology, under Contract W-31-109-Eng-38.

# Control and Supervision of a Complex Production Process Using Hybrid Systems Techniques

Humberto E. Garcia
*Argonne National Laboratory*
P.O. Box 2528, Bld. 713
Idaho Falls, ID 83403-2528

## Abstract

*New processing activities for the decommissioning of the Experimental Breeder Reactor II are being carried out at Argonne National Laboratory. The task addressed in this paper is a process to convert metallic sodium to sodium carbonate. The main idea is to characterize this sodium operation as a system that integrates real-time continuous and discrete-event components and then apply hybrid system techniques to design and implement the control and supervisory policies. This paper introduces the research in progress at ANL on this conversion process, the flow of material, and the hybrid control solution.*

## I. Introduction

The Experimental Breeder Reactor-II (EBR-II) is a liquid metal reactor operated by Argonne National Laboratory (ANL). In 1994, it was shutdown and defueling was started. Part of the planned decommissioning activities includes disposal of the sodium used as coolant for the EBR-II and FERMI reactors. The plan is to convert the chemically reactive radioactive sodium coolant to sodium carbonate, which is a chemically inert form suitable for near-surface burial as a low level waste. The schedule goal is to convert approximately 660,000 liters of sodium. This goal demands a system with high reliability, productivity and safety.

## II. Operations for Sodium Processing.

The chemical conversion process will be conducted in the Sodium Process Facility (SPF) at ANL-West. The conversion will be performed in two steps. The first step, which converts sodium to sodium hydroxide, will be conducted at the SPF Sodium Hydroxide Process System (SHPS). In particular, SHPS will transfer sodium stored on site to a reaction vessel, combine it with water in the presence of sodium hydroxide to produce caustic, and deliver the caustic for the following step. As seen in Figure 1, sodium is first moved from two different sources to the Sodium Storage Tank (SST). The first source of sodium comes from about 1400 208 liter barrels that contain FERMI sodium. The second source is the Secondary Sodium Storage Tank containing EBR-II sodium that has been transferred there by a piping system. With sodium in the SST, it is first moved to the Day Tanks (DT) for reaction. The sodium at a DT is then transferred to the Reaction Vessel (RV) where it is combined with water in the presence of caustic to form additional caustic. The resulting caustic is moved from the RV to the Caustic Cooling Tank (CCT). Caustic from the CCT is pumped out and used in the next conversion process. This second step, which converts sodium hydroxide to sodium carbonate, will be performed at the SPF Sodium Carbonate Process System (SCPS). As seen in Figure 2, the sodium hydroxide is pumped out from the CCT(at SHPS) and fed to a horizontal thin-film evaporator blanketed with a carbon dioxide atmosphere. Here, the caustic reacts with the carbon dioxide to form sodium carbonate. The reaction is exothermic and produces most of the heat necessary to evaporate the excess water, leaving a dry sodium carbonate product. The evaporated water is condensed and drained to a water holding tank, and the carbonate is discharged into drums for landfill disposal.

## III. Process Control System.

SPF will be monitored, controlled, and supervised by a computer system. The sense, control, and operator interface functions for each process will be incorporated into a distributed control system consisting of a control and an input and output (I/O) front-end computer as seen in Figure 3. All of the process sensor and actuator wiring comes into the I/O front-end (STD Bus based) computer via terminal strips connected to the I/O boards. The front-end computer communicates with the control computer via a serial link. The control computer drives a graphic terminal used to monitor and control the SPF systems. These two computers contain all the software needed to read the

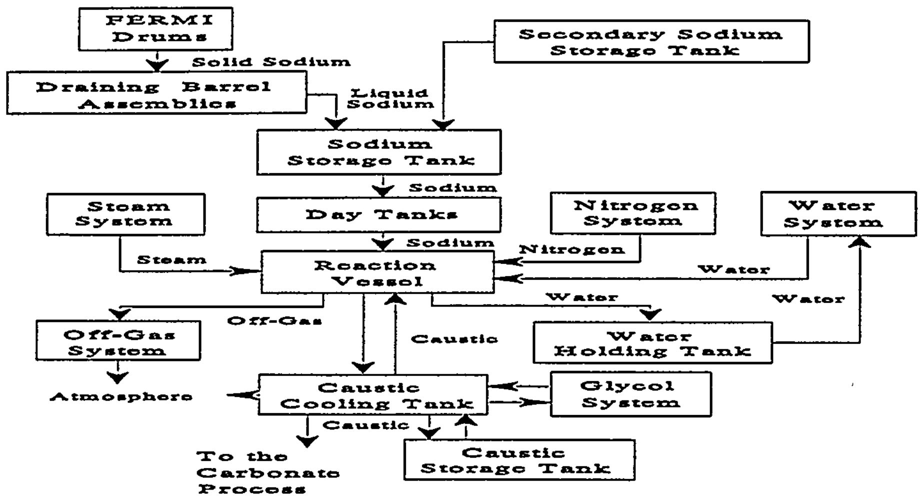

Figure 1. Sodium Hydroxide Process System (SHPS).

process sensors, to drive process actuators, to condition data, to control the SPF through its various operating modes, and to display process graphics and interact with the operator. To this end, the major development efforts include: 1) Development of a hybrid model of the sodium processing operation and 2) Utilization of hybrid system analysis techniques to characterize and validate the behavior of the actual process under the control of a supervisory agent. Specifically, this industrial system is hybrid by nature consisting of continuous time processes controlled by discrete-event processes running on real-time computers. In principle, the integration of real-time continuous and discrete-event components into a control system can be done in an ad hoc manner. However, feasible operational states, changes and conditions may be far more difficult to categorize, correctly interpret, and respond to if a formal approach is not used. Software engineering methods are being utilized in the design and implementation of the SPF control system to improve the tasks of verifying software correctness, debugging, updating, and maintenance.

The proposed system architecture is a hierarchical configuration primarily built from three types of components as seen in Figure 4: 1) low-level objects, 2) high-level objects, and 3) inter-level interfaces. The hierarchy is structured in such a way that intelligence increases while precision decreases as one moves from the bottom to the top. The flow of information is well defined in that communication is only possible among directly adjacent (above/below) objects. For example, the scheduler cannot directly communicate to a given supervisor without first going through the coordinator. Similarly, information flow among objects at a same level is kept to a minimum. In case information from a given object is required by another, their higher supervisory object would serve as their communications mediator. The aim of these information flow design criteria is to increase the robustness of the final code. With respect to the components of the proposed architecture, the low level objects correspond to the low level controllers that interact directly with the physical process. Their control actions, which modify manipulated variables, result from measured process variables and commands received from higher level objects. Four types of high-level objects are identified, namely, the *scheduler*, the *coordinator, supervisors*, and high-level *controllers*. In particular, the scheduler interacts with the user to define the set of concurrent activities (to be defined later) to be executed at any time. The coordinator then directs and coordinates its subordinate supervisors in such a way to complete the set of commanded activities. Supervisors are

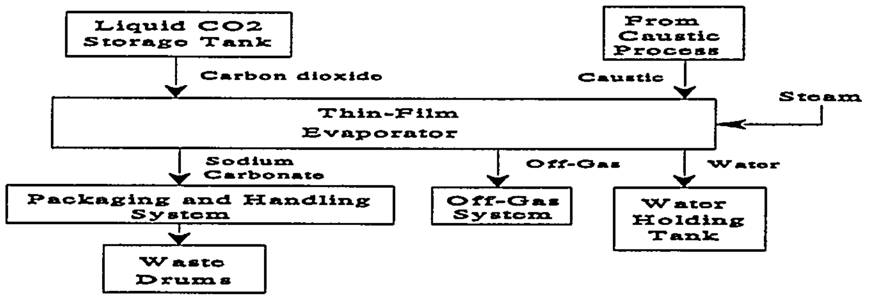

Figure 2. Sodium Carbonate Process Systems.

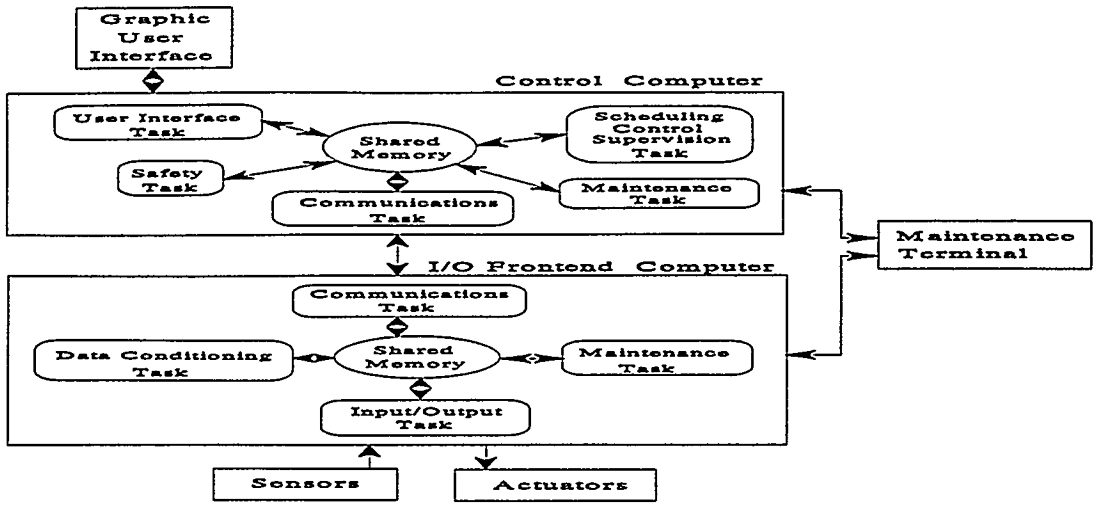

Figure 3. Sodium Process Facility Computer Control Systems.

then defined based on functional and physical partitioning. Each supervisor has a defined range of possible action and responsibilities for configuring its underline control scheme in such a way that every required activity is supported. For example, based on Figures 2 and 3, instances of implemented supervisors include the DrainBarrel Supervisor, the Reaction Supervisor, the DayTanks Supervisor, the CausticStorage Supervisor, the Carbonate Supervisor and the Support Supervisor. For instance, the DrainBarrel Supervisor is responsible in assuring that its assigned end-effectors, i.e., sensors and actuators, operate properly in support of each activity it may be involved in at any time. Depending upon the underlying complexity of the supervisory task, a supervisor may rely on subordinate supervisors or *sub-supervisors* to accomplish the assigned activities. For example, the DayTank Supervisor relies on two sub-supervisors, each one defined for one of the existing two day tanks in the plant. Finally, the objects at the lowest level of the hierarchy shown in Figure 4 correspond to the high-level controllers, which gather information on the current operational conditions of the controlled process and direct commands to the low-level controllers to govern the behavior of the physical process.

As mentioned before, the desired behavior of the plant is given in the form of a set of required (concurrent) activities that should be performed by the system. This activity set is enforced by the scheduler which verifies that this set is consistent in the sense that all listed activities can be executed concurrently without violating safety and performance constraints. In particular, an activity is defined as a *task* in a given *mode*. A task specifies a given operation. Instances of tasks include DrainSodium, ReactSodium, StoreCaustic and MakeCarbonate. For example, as seen in Figure 1, the task DrainSodium defines all the operations required for extracting the sodium contained in a number of drums (FERMI Drums) and transferring it to the Sodium Storage Tank. On the other hand, a mode identifies the operational status of a given task. Examples of modes are Shutdown, ColdStandby, HotStandby and Run. For example, a task in mode ColdStandby defines all the setup conditions that must be met in order to maintain the tanks associated with the given activity at a specified operating temperature. Similarly, in HotStandby, not only the associated tanks but also valves, pumps and piping are taken to proper operational conditions. Thus, a task (e.g., StoreCaustic), in a mode (e.g,

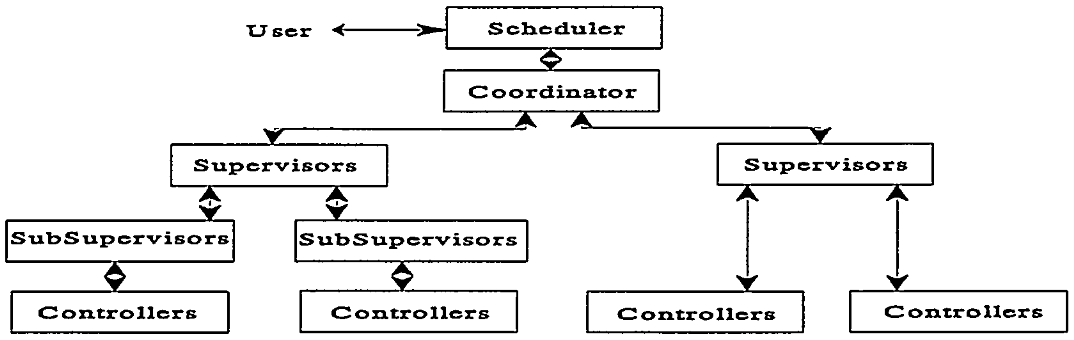

Figure 4. Hierarchy of the Control and Supervisory System.

HotStandby), defines an activity (e.g., StoreCaustic.HotStandby).

An activity may require the participation of one or more supervisors in order to be executed. The set of supervisors involved in supporting a given activity is called the *Activity Supervisors Set*. Each of these supervisors is responsible for assuring that its end-effectors are operated in support to the commanded activity. For example, to execute the activity "MakeCarbonate.Run," the CausticStorage, the Carbonate and the Support supervisors are needed; however, the DayTank supervisor is not involved in this case. Supervisors could have also been defined in relation to the completion of activities, with only one supervisor being responsible for each activity. Because of the possibility of concurrent execution of activities, this approach could result in a controller receiving commands from several supervisors whenever its end-effectors are involved in executing more than one activity. To resolve these multiple supervisory order situations, it would have been required to provide controllers with decision-making capabilities. This would have somewhat violated the principle of decreasing intelligence while increasing precision when moving down in the proposed hierarchy and potentially complicated the final implementation. Partitioning supervisors based on underlying physical boundaries assures that only one supervisor can govern the operation of any given controller and places the coordinating functions on higher level objects.

To assert the current operational phase or condition of the given activity, *status* is introduced. Possible activity status includes Idle, Setting, Ready, Working, Suspend and Down. For instance, an activity in a "Suspend" status will indicate that it has been suspended momentarily in order to resolve an abnormality in the system. The status of each activity is used by higher modules to coordinate the operation of their subordinate objects. For example, to compute the status of a given activity, the coordinator looks at the statuses reported by each of its supervisors involved in supporting the activity. Based on the reported statuses (at the supervisory level) and given guidelines, the coordinator assigns the status for the activity as seen at its level. For instance, assume that a given activity involves three supervisors. Two supervisors indicate to their coordinator that the commanded activity is Ready to be executed while the third supervisor reports the activity as still being Setting at its domain. The coordinator thus reports to the scheduler that the activity as still being in a Setting status; the activity will be declared Ready to be executed when all supervisors indicate so. This activity status (at the coordinator level) is then utilized by the scheduler to resolve operational inquires. Similarly, if an actively involved supervisor relies on a set of subordinated sub-supervisors to support a commanded activity, it asserts the activity status by gathering the reports communicated by its subordinates. This procedure extends all the way to the control level where no further division of operation occurs. Controllers, which are the only objects in direct contact with the physical system, finally retrieve the information required to assert activity statuses at higher levels.

To ease implementation and verification of the correctness of the control software, a real-time object oriented approach is being utilized. Object-oriented concepts such as inheritance and class abstraction can be beneficially employed in the design of real-time systems to enhance reusability, understandability and software quality. In addition, formal modeling approaches are being used to clearly characterize the behavior of objects. In particular, each activity executed by a given object is modeled as a *hybrid mechanism*. A hybrid mechanism [3] is here defined as a tuple M: $(\Gamma,\Psi)$ where $\Gamma$ and $\Psi$ are the *static* and the *dynamic* components of M, respectively. The static component is the tuple $\Gamma$ : $(N,\Sigma,T,X,P)$ with N denoting the set of *nodes*, $\Sigma$: set of possible *events*, T: a tuple of *timers*, X: a tuple of *state variables*, and P: collection of *state predicates*. The set $\Sigma$ is further divided into *controllable* events (e.g., actions) and *uncontrollable* events (e.g., environmental responses). On the other hand, the dynamic component is the tuple $\Psi$: (d,f,h,I) with d: *possible events* function, f: the *state transition* function, g: the internal *evolution* function, h: associated *constraints* function, and I: initial conditions. Each object verifies that its set of current activities is consistent, with their progress following this hybrid model. In particular, an activity being in a given node precisely defines the operational conditions and possible responses that could be observed to occur in the event of operational changes. To specify the requirements, real-time logic techniques are being investigated [1, 2]. It is expected that the limited and similar responsibility attached to each object and the formal modeling strategy characterizing their behaviors will increase the overall quality of the implemented software. Further discussion of the system will be given in subsequent papers.

## V. Acknowledgments

The author would like to acknowledge the assistance offered by the staff at ANL-West and, in particular, the efforts of R. Carlson, L. Christensen, D. Cutforth, C. Livengood, S. Start, and R. Washburn.

## References

1. Hybrid Systems, Lecture Notes in Computer Science 736, Springer-Verlag, the Netherlands, 1993.
2. Real-Time: Theory in Practice, Lecture Notes in Computer Science , Springer-Verlag, 1991.
3. H.E. Garcia, et al, *A Reconfigurable Hybrid System and Its Application to Power Plant Control*, IEEE Trans. on Control Systems Technology, Vol. 3, No. 2, June 1995.