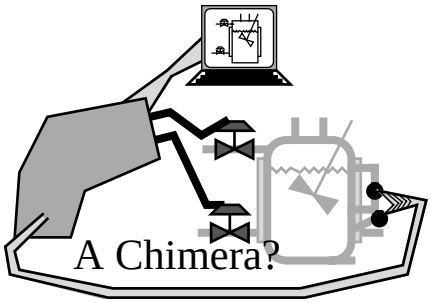
Deriving the Human Interface from the Automatic Controls

E. H. Bristol
The Foxboro Co.
Dept. 0318, Bldg. C41–1H
Foxboro MA, 02035
Tel. (508) 549-2019
Fax: (508) 549-4379
email: ebristol@foxboro.com



Abstract

Any significant control system is intended to implement a well defined set of tasks in cooperation with some operator, making human interfacing central to the design of effective controls. Applications designed by control engineers alone can easily fail for lack of effective human interfacing. But control and human interfacing have different traditions, the one based on a primarily mathematical theory, the other on scenarios, formalized testing, social science, and visual or psychological design. Modern control requires more systematic treatment of the paired technologies, in order to better guide the interface design, and support the economic implementation of combined system designs. Such treatment would also allow interface designs which adapted to dynamic changes in the application situation. The paper will introduce the topic, discussing the role of the operator in the face of different classes of control function, setting the stage for other papers dealing with different human factors issues behind this kind of situation adaptive interfacing. It will also show how control systems and human interfaces can be implemented to parallel each other automatically, when derived from a common statement of control intent (in this case expressed in statements in a proposed computer process control language).

1.0 Introduction

Any significant control system operates under some human operator, whether he be a process operator or an airplane pilot. In an ever more competitive economic environment, the nature of the controlled task becomes more elaborate at the same time that pressures to cover more of the operation with fewer people and utilize more sophisticated technology becomes more insistent.

Critical to the accommodation of these requirements is an improved controls/human interfacing capability. As with the design of the controls, the human interfacing design is affected by the same pressures as the operating environment: Do better with less:

- Fit the interface better to the controls, as designed;
- Enable the interface to adapt to the working situation as it evolves;
- Better support the operator as he tries to diagnose abnormal situations under pressure;
- Allow operation with fewer operators per plant; and
- Do all of this with less cost, man hours, and talent.

These pressures are most naturally served by placing the human operator in a more supervisory role, over ever-more-automated controls. But there is the inevitable necessity (driven by occasional emergencies) to support him seamlessly in lower levels of operation as well.

The control system should implement a well defined set of tasks and functions in cooperation with the operator. But the control and interface design activities heretofore represent two quite different technical attitudes: Controls are based on conventional engineering, simulation and analysis. Human interfacing is based on social science and artistic thinking. As a result, control system designers are unlikely to be good in both disciplines.

The problem calls for a more formal understanding of the relation between the control system and its interface. But formalization can be more than a means for improving the design of static operating platforms, such. It can lead to human interfaces that automatically adjust to the nature of the problem that the operator is facing, providing what are called situation adaptive displays.

This paper will set the stage with a discussion of the role of the operator, paralleling Sheridan.[1] These roles are affected by the different forms of control. There are many considerations in defining situation adaptive displays, but a starting point for this paper will be to show how the display can be

derived from the design intent that the interface shares with the control system being operated.

The following papers will then address the control/interface problem from a number of other perspectives: the nature of the process, the nature of the task, the nature of the human being, and the response of the operator to different levels of information loading.

Background to the discussion are the author's ideas of intent based control programming as elaborated below.[2, 3] These ideas have expanded into a computer process control language proposal.[4-9] This paper ends with examples showing how such an intent based formalism can equally well be the basis for human interfacing as for controls.

In the process control application, the goal is automatically generated display structures. This may be the best means of achieving a good fit between controls and interfacing, under the application's complexity and design economics. Whether or not appropriate to other application classes, a greater emphasis on intent, as a formal bridge between control and interface design, is an important step forward.

Intent based control is the specification of the control in terms of standard application objectives (regulation to a specified control target, constraint to some value inside a specified bound, etc.), which can be implemented in standard well understood structures. Intent based control can be effective as the basis of a control/interfacing language only if other objectives are also achieved:

- The set of standard objectives need to be sufficiently general that the intended class of applications is completely covered by them.
- Their interconnection strategies must be as complete and seamless as with any other computational operator.
- The objectives chosen must be of sufficiently high level that the collective intent of any normal composite application design is transparent to its normal practitioners. For example, it is well known that all computations can be reduced to the combinatorial application of **AND**/**OR**/**NOT** operators; but these operators are clearly at too low a level to convey the intent of normal computations.

As with any other formalism, this requires the development of appropriate primitives (in this case intent primitives):

- Some primitives will be conventional. Thus arithmetic operators in a process control context normally define simple nonlinear compensation; there is thus no ambiguity of intent in their usage, or significant need to relate their function to the human operator.[1]
- Primitives may also involve a different kind of data than normally used. Thus boolean variables obscure intent in a program; naturally named States provide a much clearer vehicle for logical controls if combined with an appropriate computational model.
- Particular intent operators may be needed. The Idiom control operator discussed later was developed because the conventional control primitives were ambiguous as expressions of intent.[2]
- In this respect, it will sometimes be necessary to define sets of intent primitives whose control implementation may be similar, even as their intent and human interfacing is explicitly quite different.

To work, each primitive must be naturally associated with a standard objective, as defined above. It must have a natural control implementation. It must have a natural interface implementation.

Some caveats: Usual computer human interfacing discussions study the interfacing for a particular application, based on Use Cases, scenarios, or scripts.[10] Or they focus on quantitative studies of particular graphic layouts, or on the definition and management of application independent graphic interface building tools, e.g. the Seeheim model.[11] The paper also does not address the computer science of actually compiling these displays.

Instead the discussion focuses on the mapping the intent to controls and their interface. It addresses the information requirements of the application and its interface. It addresses the basic requirements of automatic generation of the displays, not the quantitative issues of detailed graphic interface design.[3]

---

[1] This language also uses footnotes to express low level compensation computations whose inclusion in the body of the control program would otherwise obscure the flow of clear intent. The analogous practice will be followed in the flow of discussion of this paper.

[2] For example, the PID controller can be used to implement five or ten distinct kinds of regulation or constraint override function. In an intent formalism, these separate functions would correspond to different primitives.

[3] But it does permit a detailed graphic design effort to focus on generalized primitives applicable to a class of applications rather than single ones.

For example, simple controllers were traditionally supported with simple faceplates, which showed the target value, the actual value, the output to some manipulated variable, and various control mode states. This provided the operator with all of the external information available to the controller to support its task; it sufficed to allow the operator to intervene in that task when needed.

It might be argued that the operator would be better supported by recordings showing the history or trends of the variables under his control. This is derived data which is in fact useful. But for the purposes of our discussion, the differences will be a matter of "Policy"; both forms of interfacing are usable at some level.

Thus, for every involved class of control activity, the discussion will focus on:

- the basic control information,
- the related information that the operator should have at hand to monitor the operation of the controls,
- information supporting necessary intervention, and
- information necessary to understand its interactions with other such activities.

Put differently, the discussion addresses the higher level information involved in relating the controls to the interface, or in implementing the kind of control interface integrating language proposed, rather that in addressing the "algorithmic" level of detail for either control or operator intervention. This is the information that tells the operator what the controls are doing[4] and what they use to do it.

2.0 The Role of the Operator

The text by Sheridan[1] develops an extensive taxonomy and model of the function of the human operator of a control system, integrating the work of a number of authors. The following discussion parallels his discussion in simplified form. As a general principle, the control system operator has a number of distinct roles:

- The activities automated by any control system each have natural targets or activation conditions, or other high level means of specifying intended operation. In this normal operation, the goal of the operator is to impose the desired quality, mode, and timing of operation, changing targets, or activating different parts of the automation or control program to meet these goals. This targeting may call for continuous variation as with traditional manual control (e.g. the pilot flying an aircraft as he continuously directs the action of the various flight surface servos). But in an industrial context, where an operator may have several hundred control loops under his responsibility, it takes a more supervisory form. It will also address higher level choices such as selection of entire recipes.
- General process and control state monitoring, scanning or querying the current process state and ensuring normal operation.
- Recognition and diagnosis of abnormal operation. This splits into the recognition and proper servicing of failures whose behavior is well understood, and the diagnosis of situations with no established practice. In principle, the first situation could be covered by appropriate automation. But this may be too costly or too specialized to be practical. In that case, the human interface may be augmented with the capability to recognize patterns of events, calling for failure corrective action. True diagnosis or debugging of unforeseen situations require automation based on general principles for recognizing relevant groupings of the problem symptoms and relationships which might be important, independent of any presumed process causal model.
- Intervention in the control system to save an abnormal situation. Once a situation has been recognized it requires the operator action to restore normal operation. This may require natural use of normal controls, intended operation of emergency controls, or intervention in the normal controls to support innovative operation to overcome failures in the normal actuator degree-of-freedom path, like an aircraft pilot using engines to control direction or altitude. In any case the role of the operator as the controller of last resort is a critical consideration. In an aircraft, lives are at stake. The same may be true in the industrial process. But even when this is not a consideration, the billion dollar plant places an incentive on continued operation, even if the operator must take over normally automated functions.[5] This requires that the normal higher level automation allow seamless lower level intervention. It may involve organized access to any sensor, target, or actuator data used at any level in the control system.

[4] This is meant in the general sense. Except when the operator is actually intervening in a control loop he is not interested in the detailed values. He may be interested in how well the control is meeting its targets. He is certainly interested on whether it is operating in its normal mode, or is somehow being overridden.

[5] **Any time** an operator intervenes in the automation it to execute some alternative task variation not served by the automation.

2.1 Control Functional Classes

Operator activity pursued in support of these roles depends on the nature of the controls being operated. The following discussion will depend on a classification of controls:

- Continuous closed loop controls, oriented toward the maintenance of the constant or controlled state of process as defined by its real valued (analog) process variables. As later developed, control design intentions and corresponding operator interventions are centered about control Degrees of Freedom.

  Thus the normal control of direction of an aircraft may be controlled first by the pilot driving the controls, which transmits signals to the flight surface actuator, which affects the flight surface position, which causes the aircraft to change direction.

  In a working aircraft, any or all of these steps in the causality or Degree of Freedom path may be implemented in a closed loop control or servo. When completely automated, the Degree of Freedom path defines a sequence of cascaded controllers.

  The Degree of Freedom path is marked by the cascade controlled measurements and the final actuator. The control or manual operation of the aircraft may, in principle, involve automated overrides or manual intervention at any stage of this path, breaking the path. And in the industrial counterpart context it frequently does.

  Particularly in the industrial context, the Degrees of Freedom paths define the natural display groupings of targets, sensors, and corresponding manipulated values. They also define the basis for recognizing overrides and limits to a given control, as path changes or switchings.

- Traditional manual control, of the sort carried out by the pilot flying the plane involves the pilot directly in control loops; it requires a human interface in which the process behavior is presented to the operator, and the results of his control decisions are returned to the process in a closed loop. As quoted in the above reference, such activity depends on skill based behavior.[1, 12]

  In the industrial context, the process response is slow enough that the manual control is a matter of cogitated action rather than reflex. Also the dedicated operator operation of a single loop out of many under his supervision is impractical. In this case, manual operation needs further interface assistance in the form of alarms or annunciators which inform the operator when his attention is needed.[6]

- Logical or State oriented controls, oriented to the operation and protection of discrete state process elements: the motors and pumps. Logical control is a form of time continuous control, but without well developed Degree of Freedom traditions. In part this is because logical control does not require the use of feedback or multiple levels of cascading to achieve robust control.

  Except for its many interlocks it is largely open loop based.[7] Nevertheless, logical controls show up in many isolated forms at low levels, associated with the operation of motors, or at high levels, associated with controlling the different operating regimes. They may represent a dominant percentage of the actual controls.

  The design intent is best reflected in some form of state diagram. In this form it frequently reflects an underlying sequenced operation. State oriented controls often are used to implement sequenced controls as below. But they should more properly represent situations in which a number of alternative paths or cycles are possible. The resulting sequences are then an incidental consequence of higher system goals rather than the expression of them.

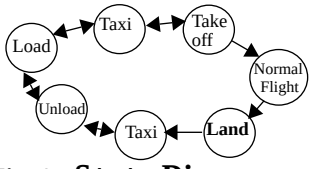


Fig. 1. State Diagram

- Sequenced operation of production, either in the form expressed as programmed phases or trajectories of batch processing, or the startup and shutdown or grade changes of continuous control. Sequencing is an intuitive form of control from point of view of intent and operator interfacing. It defines explicit steps which become the basis of any operating intervention, permitting selection of out of sequence execution.
- Theme Oriented Controls (illustrated later) in which the components are coordinated under some specialized theme. This theme is the basis of special display functions and operation.

Table 1 stages the following discussion, initially suggesting how these Operator Roles might interact with the Functional Classes. As a general principle, the operator is allowed to intervene at any point in a function where one intent primitive might take an action on another or be succeeded by another. The structure of each set of intent primitives provides the natural points of intervention.

---

[6] The process control operator may be responsible for several hundred control loops any one of which may, under some circumstances, require his attention. As his attention is shared between this number of loops, he cannot be expected to focus on more than one at a time. The automation must call on him for manual control rarely, making as little demand on his direct control abilities as possible.

[7] More formal Degree of Freedom emphasis would benefit by supporting the generation of clear displays of interlock behavior.

The operator is given the freedom to so intervene, because his requirement presumably transcends activities already automated.

2.2 Special Roles

The operator described in the main section is implicitly the dedicated operator involved in the normal productive role of the control system: the process control operator or the airline pilot. But there are special roles, requiring more engineering capability and appropriate human interfacing: the pilot plant development of the process, the initial operational design debugging, or process startup. This effort and interface is necessarily more complex, with a capability for looking at more different kinds of information.

For some kinds of processing situation, these kinds of activities may be required at all stages in the productive life of the system. They involve the same basic activities as described above, but may involve especially experienced or trained operators, and special flexibility in the on line programmability of the system.[8]

2.3 Other Interface Design Considerations

In addition to task based function there is a need for organizational structures which control access to task specific function. These structures group functions for convenient reference and access. The most common organizational structure is the hierarchy.

The traditional industrial interface was a large static panel. It still has the advantage of providing the operator with the simplest access to his data: just looking at it. But in the computer/monitor world this is replaced by a broad hierarchy of access. Here the operator follows the natural hierarchical organization of the plant down to the level at which current task data is accessible, using some menu or mouse pick mechanism. The controls and interface are divided similarly under this organization.

Sometimes considerations of task and organization may be combined: An interface may use the natural structure provided by the basic set of tasks as a model through which the necessary elements for carrying out other, possibly unrelated, tasks can be accessed. Any large system will require operator

[8] This principle is not as completely separated from the usual interface design task scripting approach as might appear. /rather, it assumes that all application requirements have been factored into the choice of sensors, actuators, and control primitives beforehand.

Table 1: Support for Operator Roles vs. Functional Classes

Operator Role / Functional Class	Normal Mode and Target Selection	Process and Controls State Monitoring	Abnormal Operation Recognition/ Diagnosis	Abnormal Operations Intervention
Continuous Controls	With Conventional Controller Faceplate Setpoints and Mode Switches	Faceplate Indicators, Trend Records	Integrated Alarm Displays, Expert Systems	Manual or Cooperative Intervention
Manual Controls	Operator Action	Operator Tracking	Direct Operator Recognition (e.g. a pilot recognizing a Stall)	Special Operational Procedures (e.g. Stall Recovery)
State Oriented Controls	Start Buttons/Choice Switches	Panel Lights. Logs	Alarm Lights	Shutdown/ReStart Logic
Sequenced Control	Task/Recipe Selection/ Parameterization	Trend Records, Progress Indicators	Alarm Displays	Manual Operation, Alternate Task Selection, Skipping or Repeating Steps
Theme Controls	Activation, Parameterization	Trend Records, Tracking Displays	Alarm Displays	For sequenced function: advancing or repeating internal clocking. (As with a Washing Machine Timer)

activities, as outlined above, which extend beyond the interfacing functions explicitly designed into the system. Thus all such elements must be designed with flexibility and inclusiveness.

3.0 Process Control Problem

Process Control traditionally focuses on low cost, one of a kind designs, not compatible with elaborate control or human interface studies. As a result, the development of control implementation vacillates between emphasis of iterated creation of simple control loops and difficult but flexible creation of general block diagram control structures. The first is simple but inadequate, the second requires a controls expert. Neither is entirely satisfactory. The graphic human interface poses the same kind of dilemma: Either the tool supports simple controls or allows flexible process graphics and requires a human factors expert (who may not understand the controls).

This contrasts with the classical single loop controller, which included its own natural faceplate. The reason that the single loop controller can be provided with its own faceplate, is that the transparent intent and corresponding function of the controller carries over simply into a corresponding intent and function of the faceplate. Thus the basis for generalization from a single loop controller perspective to a larger view is the definition of appropriate intent primitives and an architecture in which they could effectively co-operate.

As indicated earlier, the interfaces so generated should be more than a simple concatenation of faceplates one for one derived from the controllers. The resulting intent model should define intent on the basis of the individual primitives and as groups of primitives interact. The examples will show this capability to express interacting effects.

The discussion will restrict itself to the interfacing involving the normal process operation.[9] As indicated, it focuses on the information inherent in the application and necessary for effective operation, rather than on the best display ergonomics. Illustrated displays show automatically generated interfaces possible from the associated control specifications, without any attempt to be ergonomically definitive.

Classically, process control has been divided into continuous and batch or sequenced control. One of the goals of the proposed language has been to support the operational nuances of these two basic modalities in a single language based on the more detailed functional divisions described earlier. The language statement forms blend the distinct control forms seamlessly in their programmed execution. Operating displays will show this same integration, with separated displays for different classes of function.

3.1 Hierarchy Modeling

As suggested above, large system control and interface elements must be structured, usually in a process hierarchy. The language supports separate process[10] and functional[11] hierarchies. The process hierarchy divides the control system and its associated interfaces into manageably small pieces. The functional hierarchy allows different functions, within a given process division, to be separately represented for control and interfacing purposes.

Collectively, hierarchies support a number of control and human interface requirements:

- The hierarchical reference and access to control function values, targets, modes, for operator access or control connection. This carries over normal computing practice where each function is supported by a small name space. Its data is accessed hierarchically in terms

[9] The language proposal also includes interfacing characteristics designed to support engineering access and debugging on a number of levels, appropriate to the previously indicated special operating roles:

- A built-in simulation mode and capability wherein the control could be switched to a simulation and otherwise operated through all of the normal operating modes. This capability is most useful for debugging the control sequencing and logic.
- Display of the control program, reverse video-ing any active control statements (which may occur in parallel as well as in sequence).
- Recording in the control data base, for each major program variable or parameter, the point in the program which originated its most recent change in value.

[10] Expressed as programmed “Operations” and subOperations defining the controls for each process equipment.

[11] In distinctly formatted “Pages”, each representing a distinct programming form: process and program variable declarations, control Procedures, control Parameter tabulations, etc.

of this name space and the name of the function itself.

- Hierarchical operating and summary displays, presenting several levels of key process states and alarm behavior, propagated automatically according to some specification or rule from the detailed controls to the summary.

The basic, largely conventional, language hierarchy, is designed to reflect the major process unit divisions.[12] But the real world is divided into overlapping hierarchies. Thus a given process variable or function can be a member of several overlapping subdivisions simultaneously. For example, a data value may be a member of the Reactor Unit at the same time as it is a member of the Product Stream passing through that Unit. Informally these overlapping hierarchies structure the process nomenclature.

For example, the Reactor Product Feed, named for a process equipment (the Reactor), the material measured (the Product), the particular flow (the Feed). Formal intersecting hierarchies define an abstract coordinate system for locating system elements, individually or as groups. Later discussion will illustrate the benefits that this provides to the interface in automatically locating problems for the operator.

Utilizing the multiple hierarchies, the goals of the proposed automatic generation of controls and interfaces include:

- More complete higher level summary capability.

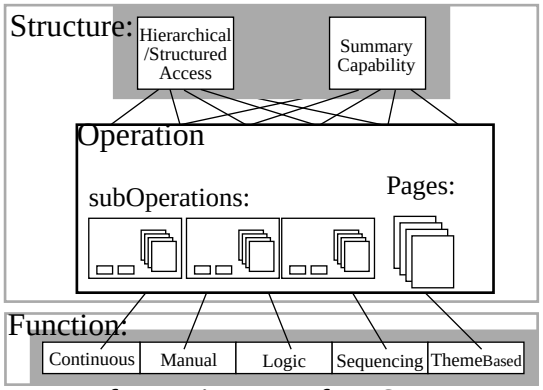


Fig. 2. Information Interface Structure

- Hierarchical access in which the local access "selection buttons" provide dynamically generated data about what will be seen when that selection is made.
- Lower level displays, accessing particular functions in a faceplate like manner, treated collectively rather than individually, so that functional interactions are clear. As elaborated later, different user preferences can be reflected in different classes of equivalent display. The operator can then chose displays according to formal display policies reflecting secondary goals.
- The ability to insert tailored process graphics in the structure at relevant points, which include any of the co-generated summary or integrated faceplate elements, so that the graphic plays well with the whole.

4.0 Function Models

This section details the distinct control activity language intent modeling strategies and resulting automatic control/display generation possibilities.

4.1 Continuous Closed Loop Control

One of the included language concepts is the Idiom. A control Idiom is a generalized control operator:

- Based on a standard continuous control role or intent,[13]
- Computing control actions applied to output Variable targets from the input and output Variable Attributes,[14]
- Deriving target and state data attributes from the Variables, without formal attribute connections, and
- Recognizing control Degree of Freedom path changes with external actions, failures, limitations, and overriding constraints.[15]

In the Idiom Loop statement:

[12] For example, into the Feed Unit, the Reactor, and the Separator.

[13] Primary Regulation or Constraint control roles, or multivariable Blending or Decoupling roles (shaping the Degrees of Freedom path), and secondary Feedforward or Compensation roles (implemented to provide consistent feedback data within their path).

[14] Among the standardized Attributes of a Variable are its measurement and setpoint. These attributes are effectively connected together to each related control computation as a result of the Idiom compilation, making them available to all such computations as needed.

[15] This structure allows individual Idioms to function autonomously. They act to support the commands or target values that they receive by deriving commands or target values to be passed to their downstream Idioms. They recognize failure of those Idioms to meet targets, adjusting their own operation to achieve the best alternative result. This generalizes many standard practices such as anti-windup and blend pacing. It allows controllers to continue to operate properly without regard to the action of other controller's occasionally conflicting objectives. Excepting the niceties of controller tuning, each controller can thus be positioned and commissioned independent of interactions with its neighbors.

T100AV$_{\textbf{REGULATE}}$ **P100**$_{\textbf{HICONSTR}}$ **F100**$_{\textbf{REGULATE}}$**V100**, the temperature **T100AV**, pressure **P100**, and flow **F100** are measured variables, and **V100** is a valve; **REGULATE** and **HICONSTR** are regulating and constraining Idiom operators (implemented with some desired controller form). The statement expresses the cascaded control of an averaged temperature variable, **T100AV**, following a normal Degree of Freedom path: **T100AV** is controlled by manipulating **F100** (through its setpoint), which is in turn controlled by manipulating **V100**.

P100$_{\textbf{HICONSTR}}$ represents a high constraint override. The pressure variable **P100** is monitored to be within its upper limit, otherwise overriding the control path of **T100AV** through **F100**. From a language design perspective the **HICONSTR** operator is of higher syntactic precedence, inserted into the basic Degree of Freedom path.

In addition to expressing the normal flow of control action, the statement (as with any control cascade) defines the natural points of operator intervention: This process could be controlled by direct operator intervention, overriding higher level controllers at any of the process variables. The operator could control by manipulating the setpoints of the **T100AV** or **F100** controllers or by manipulating the valve directly, depending on the operating requirements.

The language papers use an example where a convoluted block diagram, including close to 50 blocks and more than 50 intertwined connections of unclear purpose and function. This system is reduced to a set of 10 Loop Statements, each delineating the control of one primary process variable through a with a single valve through a direct Degree of Freedom path. Any constraints or overriding conditions to the path are clearly spelled out.

Informally, the result is converting a diagram, which would take an afternoon of discussion to arrive at an uncertain understanding, into a description which can be conveyed in five minutes. This benefit carries over into the clear mapping of the control intent to an interface display.

Conventionally, process control displays are based on collected individual controller faceplates. Figure 3 shows two possible operator displays, de-

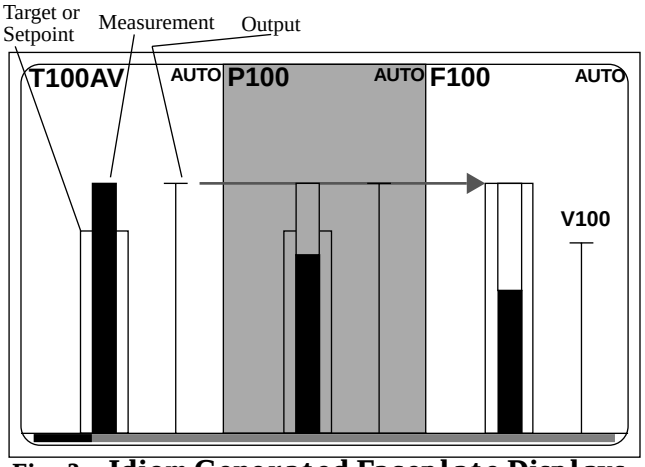


Fig. 3a. Idiom Generated Faceplate Displays

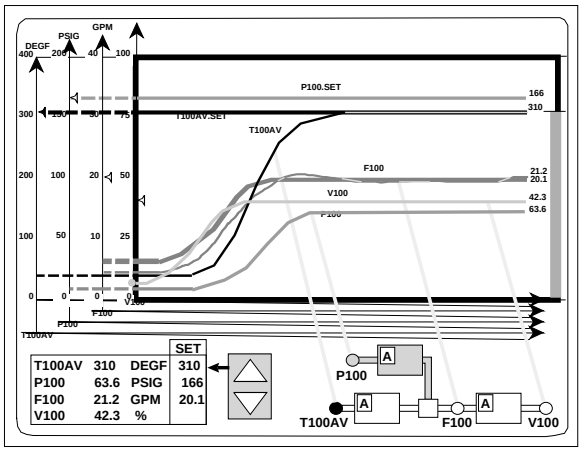


Fig. 3b. Idiom Generated Trend Displays

rived from different design Policies (the first based on faceplates, the second on trend records), that might be generated from the original Idiom statement, automatically taking into account the collective intent and number of elements. Each display would provide mouse/joystick access to the operator, permitting him to change targets, outputs, or operating states (**AUTO/MANUAL**).

In Figure 3a, the **REGULATE** and **HICONSTR** Idiom primitives map to corresponding faceplate components. Figure 3b is mapped in a more complex way, with the diagram derived from the Idioms and trend records derived from the individual variables. The diagram is an abstraction of the normal cascaded control loop with the variable's nodes providing setpoint and sensor data appropriately.

The displays are thus constructed out of primitive display forms paralleling the Idiom primitives. Each provides access to the basic control data of the larger system by presenting the basic data of the

primitive. In each case, the current Degree of Freedom path allocation can be shown with:

- The higher level interaction of override behavior (shown by shading inactive elements).
- The bypass of control data (in Figure 3a) through the inactive (shaded) elements, and
- Indication of the point of currently selected operator intervention (by underlying tick mark in Figure 3a, or the blackened variable circle in Figure 3b).

The two display Policies represent different expert preferences, but they also different task focus levels. The faceplates emphasize the current state. The trends allow more historical perspective.

The language references all controls and related parameters in terms of the most immediately associated variable.[16] Any specific display reference to process variable names can similarly access the related control displays. The language grouping of a set of controlled variables by name should be sufficient to generate the corresponding group of control loop displays, generated from composite faceplates.

With respect to situation adaptive displays, a simple loop specification statement has an additional benefit. It can be included in conditional statements which allow the control structures to be changed dynamically with changing situation. The operating displays can be dynamically regenerated to match.

4.2 Manual or Operator Controls

Operator controls reflect a continuous control perspective carried out by the operators. They will thus be subject to the same Degree of Freedom monitoring as automatic controls, but the displays should better support the operator information need, telling him where he is and where he needs to go. Within the language these tasks could be notated as special control Idioms and thereby integrated with the automatic controls. Their implementation would instead be dependent on the effective human interface.

These activities would be supported by similar faceplate displays and distinct alarms announcing significant deviations from target for action, and allowing him to pursue other tasks when not needed. This concept of operator control, in fact, provides a rationalization for that class of alarms that correspond to such well defined tasks. Similar displays are in fact needed when an operator intervenes in an automated loop.

4.3 Logic Control

Logical controls operate with continuous time data like continuous control except that they do not use feedback control. A motor once turned on is assumed to be on without checking. Where the automation includes a feedback (e.g. a centrifugal switch) its role is not to support the control of the motor but to indicate some failure of the motor or controls to act normally, and protect the motor from its consequences.

Logical controls have no Degree of Freedom tradition, but they could be presented grouping data about such a path. This is particularly true if the logic is controlling a large process “engine” which is supported by many local “motors”. As with the continuous controls, the operator could be allowed to intervene at any level necessary to accomplish the job.

The language replaces all boolean quantities with meaningfully named States.[17] Both the control designer and the operator should be able to deal with the logical states of the process in process related terms: **ON/OFF**, **START/STOP**, grouped naturally.

The language includes special variables for representing these States as well as Time and Counts quantities which typically participate in logical controls. All process variables need to be supported with additional operational attributes which support better operational interfacing:

- Input/Output connection data.
- Status data for variables whose associated sensors can become inactive.
- Target values for control, timing, and counting variables.
- Alarm states and limits.

A combined controls/interface design system

[16] The parameters associated with the **T100AV$_{\text{REGULATE}}$** Regulate Idiom phrase are referenced as attributes of **T100AV**, etc.

[17] But includes the capability to map these States arbitrarily into any desired fields in hardware communicated data.

should include widgets for displaying each of these types of data. Automatically generated operator displays should also group related State data, and distinguish between operator set States, and feedback/interlock data, in a uniform way. The result would be a poor man's counterpart of the more optimized displays illustrated under the Theme Statement discussion.

4.4 Sequenced Control

Sequencing is also important, as a way of imposing control trajectories for startup, shutdown, batch production, and various cleaning and recycling operations. Both in a language and in the operation, the most important capability is to achieve a seamless integration between the more familiar (to ACC attendees) continuous control and the sequencing which may direct or support it.

Sequencing comes in many flavors:

- Implicit sequencing as a consequence of the state transitions in a control system. Barring special human interface design, the system has no "understanding" of such a sequence or its purpose. Display takes the form of indicators for the given states, or trend records of the associated variables (as in Figure 3a).
- Lock step sequencing driven by drum sequencers or cam followers (an earlier approach to the Theme Statement control described below), where a series of events are pre-programmed to occur in a largely fixed time relationship to each other. Their operation can also be shown as a simple time based trend recording, but with the system "aware" and emphasizing significant events.
- Control against a formal multipath programmed sequence. The standard way of representing such programs is the Sequential Function Chart (SFC), marked to show progress. This can include actual transition trends.

Figure 4 shows a Batch Process. Figure 5 shows

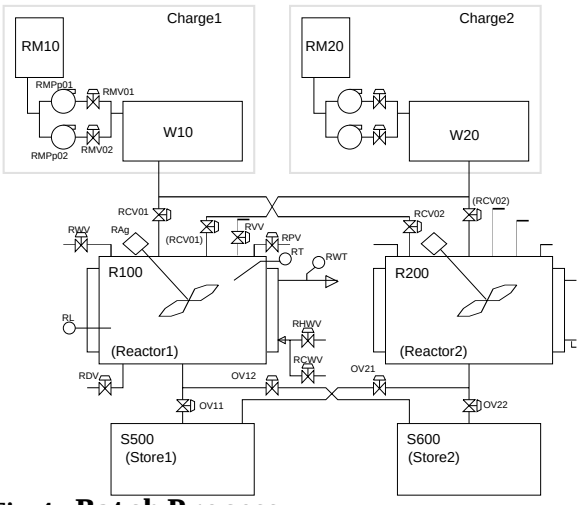


Fig. 4. Batch Process

its sequenced Recipe Procedure.[18] The language represents sequencing control structures by nested

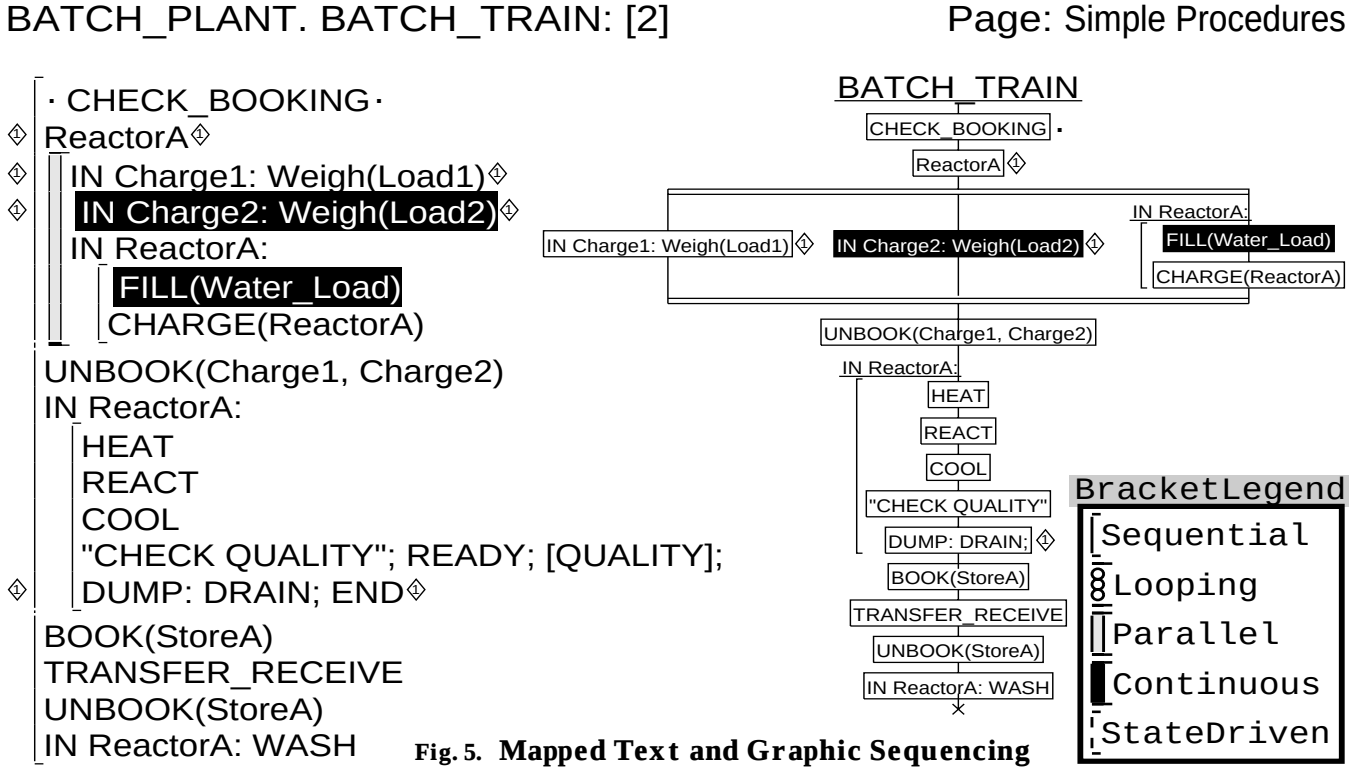


Fig. 5. Mapped Text and Graphic Sequencing

bracketed activities. The bracket shapes are keyed on different forms of execution order of their included statements. Depending on that shape, an activity may carry out its included statements, in sequence, in parallel, continuously, in a loop, or dependent on some local state. This formal structuring facilitates the automatic generation of an illustrating SFC. Operated live, currently active statements are reversed video-ed in text and graphic. Operational SFC interfaces can work similarly.

4.5 Theme Statements

Figure 6 shows a Ramp Theme statement defining a time profile of a main variable (T100) driven to the stated profile (or trajectory).[19] This structure could define an open ended number of profile segments and actions. Each profile phrase includes an optional State name (italicized) allowing the user to identify the statement phases in operational displays, and permitting the operator to stop a particular phase, or move to a different one, by name.

```
RAMP  T101   START  FOR VENTTIME
             HEAT   TO COOKTEMP + BIAS, IN HEATTIME MI
             COOK   AT COOKTEMP, FOR COOKTIME MIN
             COOL   TO COOLTEMP, IN COOLTIME MIN
```

Fig. 6. Ramp Theme Statement

Figure 7 shows two similar, policy differentiated, operating displays, for the Ramp statement. Each of the displays could be generated automatically. One emphasizes the higher level processing state, the other the trended history. The benefit, as above, is consistent, inexpensive, rule based, interfacing.

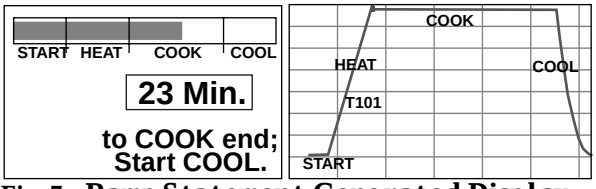


Fig. 7. Ramp Statement Generated Display

5.0 Structural Models; Categorizations and Alarms

Hierarchies structure a large system for accessing the different control displays. They are natural, easily specified, and easily generated automatically. Their normal behavior is well understood. So this section will show some unusual possibilities derived from multiple user defined hierarchies that we call Categorizations.

Alarms are a major problem for good design of any large system. Except when rationalized in support of a specific well understood operator task, as above, alarms address the poorly understood abnormal situations. Here any pre-programmed diagnosis is likely to be faulty, or a barrier to needed operator information. This problem includes many dimensions. But one of these is to give the operator the high level view of the distribution of alarms in the process in a way that provides the most control over his information, like an airplane pilot on a clear day, able to see for miles and select his level of detail, with a simple change of visual focus.

The control application vocabulary is centered about the process variables. Named Category lists can define a wide range of alarm/variable groupings, building a wider operations vocabulary which includes both the individual variable and function names and the group names. Further multiple hierarchical Categorization of these Categories could be based on:

- Process subdivisions: Reactor, Distillation tower, Chlorine plant (the traditional process hierarchy).
- Fluid streams: Product, Utilities, Steam, Lights, Heavies.
- Situations: Startup, Normal run, Heating phase, Pump problem.
- Operational Impact: Safety, Equipment jeopardy, Product quality, Production show stopper.

Each of the above Categorizations is “orthogonal” to the others. Consider just three derived applications:

- To allow the conventional hierarchical display selection organization.
- To allow the operator filtering of alarm displays. Figure 8 shows four lines, corresponding to the above Categorizations. Each line highlights one or more selected Categories, indicating an operator selection. Whichever main alarm display (Log, Trend, Summary) is selected can now be filtered to include alarms only from Categories which have been selected on each line.
- To support “One Word Summaries”. These are computed summary descriptions characterizing the alarms

[18] It also illustrates the format of one of the language Pages (the procedures Page) in simplified form designed to highlight recipe steps.

[19] The example is simplified from its original. That example makes use of capability in the statement form to control multiple variables and coordinate any number of side activities.

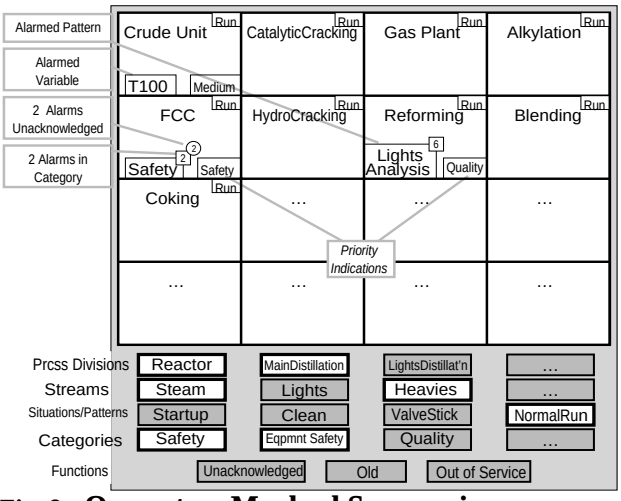


Fig. 8. Operator Masked Summaries

active in some plant unit Category. They display the name of that other Category with smallest (set) intersection with the displayed Category, still containing all active alarms in the displayed Category.

Figure 8 shows a Summary Matrix[20] for some level of alarm Category, each of whose Indicator boxes corresponds to a direct subCategory of process division summarized by the Matrix. The figure shows three examples of "One Word Summaries" each in a separate Indicator box:

1) "T100": The Crude Unit Indicator is indicating only one alarm; only its name is needed.
2) "Safety": The FCC Unit Indicator is indicating that all included alarms fall in the Safety Category.
3) "Lights Analysis": The Reforming Unit Indicator is indicating that all alarms have to do with the particular Situation Category.

The purpose of the One Word Summary is to provide the best alarm summary possible within limited information content. As a situation develops, the One Word Summary shows first the initial alarm (as with T100 in item 1), then the name of a small Category, and then the name of a larger Category, etc. As the situation develops the abstraction gets broader. This has the effect of allowing the initial detail to show up at the topmost display level. The operator data load is held constant by varying the abstraction level.

5.0 Policies

Earlier discussion suggests display Policies as a mechanism for shaping special user interface requirements in an environment when interfaces are generated automatically. There are other benefits:

- Process variables have distinct attributes. Being able to call into play different Policies which cause the grouping of these variables in different ways according to their attributes gives the operator considerable control over his display without requiring a lot of detailed display design effort. This generalizes the use of the filter rows shown earlier. The specification of these attributes at the level of their Categories rather than individually also simplifies the work. This is important in an environment, like a refinery, where there may be 50,000 variables.
- Accommodation of new concepts: the evolution of wall size screens and virtual reality greatly complicates application design. If displays could be generated by rules which implemented specified Policies, their implementation would be simplified.
- Plants are now networked like offices, but a live process control plant is real time interdependent unlike an office. Operations envision operators and task responsibilities moving from console to console to meet changing situations or operator availability. Dynamic human interfacing is dynamic resource allocation, formulated and managed by the machine under a specified set of control intents and interfacing Policies.

6.0 Summary

The discussion has suggested one way of making controls and human interfacing compatible: Derive them automatically from the same intent specifi-

[20] In the language, Operations have operating States and other higher level parameters which can be included in any associated display. In this respect they would be interfaced much like the earlier Logic Controls. They are at too high a level to have standard detailed structure of their own. Note that in the figure, alarm priority is also expressed in terms of the relevant Category names.

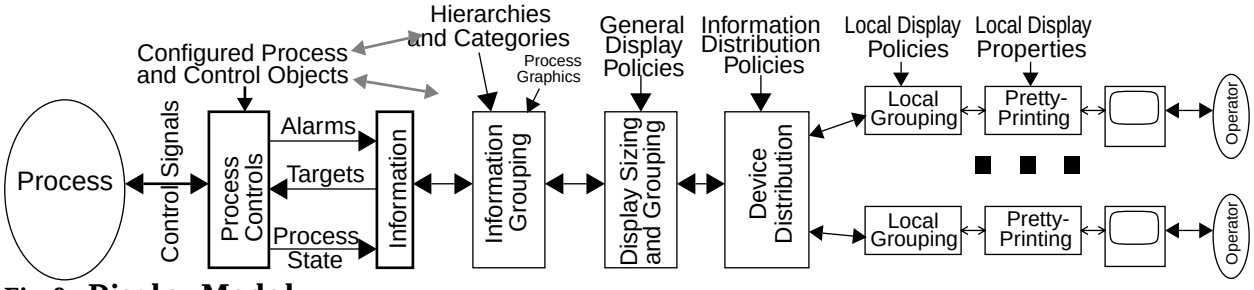


Fig. 8. Display Model

cation. This is supported by identifying distinct classes of functional operation for which the relationships and design intent of the general functions can be formalized. These formalizations can be interconnected to a hierarchical structure which supports their general access and summary. From this combination one can envision the automatic generation of an effective default user interface structure, in which tailored graphics can be inserted. In process control, the linking of he interface design to an automatic design process has the potential of improving the design by centering it about a standardized structure. There is another indirect advantage: With the advance of different kinds of multimedia, and large and small screen display technology, the creation of tailored interface designs becomes ever more onerous. An integrated control/interfacing framework like the one proposed lays the groundwork for the feasible operation in this new dynamic environment.

7.0 Bibliography

[1] T. B. Sheridan, Telerobotics, Automation, and Human Supervisory Control, MIT Press, 1992.

[2] E.H. Bristol, “Strategic Design: A Practical Chapter in a Textbook on Control”, ‘80 JACC, San Francisco, Aug. ‘80.

[3] A. Prassinious, T. McAvoy, E.H. Bristol, “A Method for the Analysis of Complex Control Schemes”, ‘82 ACC, Arlington VA, Jun. ‘82, pp. 1127–1132.

[4] E.H.Bristol, “Rules, Statements, and Idioms”, 17th Annual Advanced Control Conference, Purdue University, West Lafayette, IN, Sept. 30 – Oct. 2, ‘91.

[5] E.H. Bristol, “A Language for Integrated Process Control Application”, Retirement Symposium in Honor of Prof. Ted. Williams, Purdue University, West Lafayette, IN, Dec. 5 - 6, `94.

[6] E.H. Bristol, “Not a Batch Language; A Control Language”, World Batch Forum, San Francisco, May ‘95; also ISA Transactions, Vol. 34 (1995), pp. 387-403.

[7] E.H. Bristol, “Redesigned State Logic for an Easier to Use Control Language”, World Batch Forum, Toronto, May 13–15, ‘96; also ISA Transactions, Vol. 35 (1996), pp. 245-257.

[8] E.H. Bristol, “Batch Object Modeling and Language”, World Batch Forum, Houston, Apr. 29–30, ‘97.

[9] E.H. Bristol, “Sequencing, Logic, and Theme Statements”, Annual AIChE Meeting, Chicago, Nov. ‘91.

[10] Ivar Jacobson, Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley Pub. Co, 1994.

[11] D. R. Olsen Jr., User Interface Management Systems: Models and Algorithms, Morgan Kaufman Publishers, 1992.

[12] J. Rasmussen, “Outlines of a Hybrid Model of the Process Plant Operator”, Monitoring Behavior and Supervisory Control (edited by T. B. Sheridan and G. Johannsen), Plenum, 1976.