one. The choice of operating system, interrupt logic, scheduling priority, and system design parameters will be influenced by the latency requirements. Also, behavioral analysis of the requirements to determine consistency with process functional requirements and constraints may not be correct unless the value of this behavioral parameter is known and specified for the software. Therefore, the requirements specification must include the allowable latency factor.

- *A latency factor must be included when an output is triggered by an interval of time without a specified input and the upper bound on the interval is not a simple, observable event.*

Triggering an output on an interval of time without a specified event occurring always requires the specification of a latency factor between the end of the interval and the occurrence of the output. Where the upper bound on the interval is a simple, observable event, latency is not an issue. However, where the intent is to signal the nonoccurrence of an input after some other event, a latency specification is required.

- *Contingency action may need to be specified to handle events that occur within the latency period.*

Additional requirements may need to be specified to handle the case where an event is observed within the latency period. For example, if an action is taken based on the assumption that some input never arrived and if it is later discovered that the input actually did arrive but too late to affect the output, it may then be necessary to take corrective action.

- *A latency factor must be specified for changeable human–computer interface data displays used for critical decision making. Appropriate contingency action must be specified for data affecting the display that arrives within the latency period.*

Latency considerations also affect specification of the human–computer interface. Whenever a data display changes just prior to an operator basing a critical decision on it, the computer may need to query the operator as to whether the change was noted before action selection. The display might involve, for example, showing a set of operator options, including a recommended option and several indications of poor ones. If the arrival of asynchronous data results in a change to the recommended action, then whether the operator had sufficient opportunity to observe that change will affect the required human–computer interface behavior. As another example, an operator decision to fire a missile at a target that has just had its displayed threat value reduced (but not completely eliminated) may warrant extra interaction between the program and the operator.

- *A hysteresis delay action must be specified for human–computer interface data to allow time for meaningful human interpretation. Requirements may also be needed that state what to do if data should have been changed during the hysteresis period.*

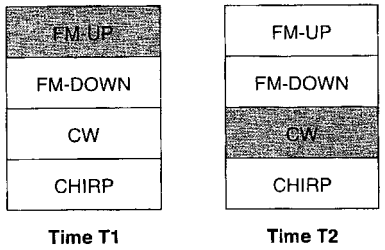

FIGURE 15.2
Two consecutive snapshots of an operation action menu with the recommended action highlighted. The recommendation must be constant long enough for meaningful human interpretation. Requirements are also needed to deal with latency problems when the recommended action changes.

Variable data, such as a computer-recommended operator action, must be constant long enough for meaningful human interpretation, which leads to a requirement for a hysteresis delay (Figure 15.2). An additional requirement will then be needed to cope with situations where the action is selected after the occurrence of an event that should have changed the displayed data but did not because it occurred before the expiration of the hysteresis delay from the previous change.

15.4.6 Output to Trigger Event Relationships

Some criteria for analyzing requirements specifications relate not to input or output specifications alone but to the relationship between them. Although, in general, the relationship depends on the control function being specified, basic process control concepts can be used to generate criteria that apply to all process control systems, such as feedback and stability requirements.

Responsiveness and spontaneity deal with the actual behavior of the controlled process and how it reacts (or does not react) to output produced by the controller. In particular, does a given output cause the process to change, and if so, is that change detectable by means of some input? Basic process control models include feedback to provide information about expected responses to changes in the manipulated variables and information about state changes caused by disturbances in the process.

- *Basic feedback loops, as defined by the process control function, must be included in the software requirements. That is, there should be an input that the software can use to detect the effect of any output on the process. The*

requirements must include appropriate checks on these inputs in order to detect internal or external failures or errors.

Feedback is a basic property of almost all process control systems: If feedback information is not used by the software, the requirements specification is probably deficient. Basic feedback loops need to be included in the software requirements, while missing feedback loops provide clues as to incompleteness in the specification.

As an example, an accident occurred when a steel plant furnace was returned to production after being shut down for repair [16]. A power supply had burned out in a digital thermometer during power-up so that the thermometer continually registered a low constant temperature. The controller, knowing it was a cold start, ordered 100 percent power to the gas tubes. The furnace should have reached operating temperature within one hour, but the computer failed to check (and thus detect) that the thermometer inputs were not increasing as they should have been. After four hours, the furnace had burned itself out, and major repairs were required.

This situation could easily have been avoided if information about the characteristics of the process had been used to predict and check for the expected behavior of the system. In this case, the only information needed to avoid the accident was that the temperature should increase if the burners are on.

If the process does not respond to an output as expected and within a given time period, there is presumably something wrong and the software should be required to act accordingly—perhaps by trying a different output, by alerting a human operator, or, at the least, by logging the abnormality for future, off-line analysis.

Ideally, process control systems should be designed such that the effects of every output affecting a manipulated variable in the system can be detected by some input provided by the feedback loop. The situation is not always that simple, however. Disturbances interfering with the process can cause changes that are not initiated by the computer or can inhibit changes that the computer has commanded.

- *Every output to which a detectable input is expected must have associated with it: (1) a requirement to handle the normal response and (2) requirements to handle a response that is missing, too late, too early, or has an unexpected value.*

Every output to which a detectable response is expected within a given time period induces at least two requirements: The "normal" response requirement and a requirement to deal with a failure of the process to produce the expected response. The failure could involve the response having an erroneous or unreasonable value, the response arriving at the wrong time, or the expected response might be missing entirely.

- *Spontaneous receipt of a nonspontaneous input must be detected and responded to as an abnormal condition.*

If the environment responds *too* quickly, coincidence rather than appropriate stimulus-response behavior may be responsible. Most processes do not react instantaneously, but only after a delay (time lag). Thus, the specification of a latency factor is required. A value-based handshake protocol can be used to eliminate the need for the latency factor: Some field of the input I identifies it as a unique response to some specific output O.

Some inputs are spontaneous—they may be triggered by environmental factors not necessarily caused by some prior output. However, an input that is supposed to be nonspontaneous (it is only supposed to arrive in response to some prior system output) induces yet another requirement to respond to a presumably erroneous (spontaneous) input.

- *Stability requirements must be specified when the process is potentially unstable.*

In addition to feedback requirements, stability requirements, such as a phase margin of at least 45 degrees and a gain margin of at least 3 decibels, may need to be specified for one or more operating states. The stability requirements apply to the process-control function, which is described by a control law or a transfer function relating output to input [23].

15.4.7 Specification of Transitions Between States

Requirements analysis may involve examining not only the triggers and outputs associated with each state and the relationship between them, but also the paths between states. These paths are uniquely defined by the sequence of trigger events along the path. Transitions between modes are particularly hazardous and susceptible to incomplete specification, and they should be carefully checked.

Reachability

- *All specified states must be reachable from the initial state.*

Informally, a state is said to be *reachable* from another state if there is a path from the first to the second. In most systems, all states must be reachable from the initial state. If a state is unreachable, there are two possibilities: (1) either the state has no function and can be eliminated from the specification, or (2) the state should be reachable and the requirements document is incorrect and must be modified accordingly.

Most state-based models include techniques for reachability analysis. In complex systems, complete reachability analysis is often impractical, but it may be possible in some cases to devise algorithms that reduce the necessary state space search by focusing on a few properties. The backward-reachability hazard analysis techniques for state machine models described in Chapter 14 are examples of algorithms that limit the amount of the reachability graph that must be

generated to get enough information to eliminate hazardous states from the requirements specification.

Recurrent Behavior

Most process control software is cyclic—it is not designed to terminate under normal operation. Its purpose is to control and monitor a physical environment; the nature of the application usually calls for it to repeat one single task continuously, to alternate between a finite set of distinct tasks, or to repeat a sequence of tasks while in a given mode. Most systems, however, include some states with noncyclic behavior such as temporary or permanent shutdown states or those where the software changes to a different operating mode.

- *Desired recurrent behavior must be part of at least one cycle. Required sequences of events must be implemented in and limited by the specified transitions.*

The specification should be analyzed to verify that desired behavior is repeatable. To be repeatable, the behavior must be part of at least one cycle, but in many cases checking this behavior alone will not be sufficient; more complex sequences of events may need to be identified. An output to turn on a piece of equipment, for example, may be inappropriate unless the last output turned the equipment off. Consider an output to start a piece of equipment. The equipment may need to be started more than once, but it could be damaged if two START commands are issued without an intermediate STOP command. To prevent this hazard, every cycle that includes a START also has to include a STOP.

- *States should not inhibit the production of later required outputs.*

An *inhibiting state* for an output is a state from which the output cannot be generated. If every state from which the output can be generated is unreachable from an inhibiting state, then the output cannot be generated again once the inhibiting state is reached. Whether or not this condition represents an incompleteness depends upon the application.

Reversibility

In a process control system, a command issued to an actuator often can be canceled or reversed by some other command or combination of commands. This capability is referred to as *reversibility*.

- *Output commands should usually be reversible.*

Outputs will usually require reversing commands. Therefore, outputs should be reviewed and classified as to their reversibility. For an ON command to be reversible, the state in which the canceling OFF command is issued must be reachable from the state in which the ON command was issued. For example, an alert condition to an operator (such as a below-minimum-safe-altitude warning to an air traffic controller) should be reversible when the condition no longer holds (the aircraft is now at a safe altitude).

- *If x is to be reversible by y, there must be a path between the state where x is issued and a state where y is issued.*

There will usually be several different classes of the reversing outputs. The appropriate reversing output, for example, may depend on whether the controller has acknowledged the receipt of the original alert, is in the process of reviewing the alert, or has taken positive action to ameliorate the alert condition. The human–computer interface in particular is full of complex classes of reversible phenomena [135].

Preemption

When the same physical resource, such as a data entry device or display, must be used in distinct multistep actions at the human–computer interface, requirements will be needed to deal with preemption logic. In addition, some actions may have to be prohibited until others are completed. An action to recompute estimated time of arrival might be prohibited until an in-progress, manual navigational update is completed or explicitly canceled.

- *Preemption requirements must be specified for any multistep transactions in conjunction with all other possible control activations.*

In general, there are three possible system responses to an operator action from a parallel-entry source prior to completion of a transaction initiated by some previous control activation: (1) normal processing in parallel with the uncompleted transaction, (2) refusal to accept the new action, and (3) preemption of the partially completed transaction.

If preemption is possible, then the attempted activation of a multistep sequence requiring the use of a resource already involved in another incomplete transaction provides the following three choices:

1. The new request could completely cancel the previous, incomplete transaction, clearing or replacing any displays associated with it.
2. The new request could preempt the shared resources, but the displayed state could be preserved and restored upon completion of the new transaction.
3. The operator could be prompted and required to indicate the disposition of the incomplete transaction, in which case there will in general be four alternatives:
   a. Cancel the incomplete transaction and start the newly requested one.
   b. Complete the old transaction and then proceed automatically with the new request.
   c. Cancel the new request and continue with the old, incomplete transaction.
   d. Defer but do not cancel the old, incomplete transaction.

If any transactions are deferred and restored, obsolete information must be identified, as discussed previously.

Path Robustness

For most safety-critical, process-control software, there are concerns beyond pure reachability: Even if a state fulfills all reachability requirements, there is still the question of the *robustness* of the path, or paths, affecting a particular state.

Consider an output that has the possible values of UP and DOWN. Suppose that every possible path from a state where an UP command is issued to any state where a DOWN command is issued includes the arrival of input *I*. Then if the computer's ability to receive *I* is ever lost (perhaps because of sensor failure), there are circumstances under which it will not be able to issue a DOWN command. Thus, the loss of the ability to receive *I* can be said to be a *soft failure mode*, since it *could* inhibit the software from providing an output with the value DOWN.

If the receipt of input *I* occurs in every path expression from *all* states that produce UP commands to *all* states that produce DOWN commands, the loss of the ability to receive *I* is now said to be a *hard failure mode*, since it *will* inhibit the software from producing a DOWN command.

- *Soft and hard failure modes should be eliminated for all hazard-reducing outputs. Hazard-increasing outputs should have both soft and hard failure modes.*

The more failure modes the requirements state machine specification has, whether soft or hard, the less robust with respect to external disturbances will be the software that is correctly built to that specification. Robustness, in this case, may conflict with safety. A fail-safe system should have no soft failure modes, much less hard ones, on paths between dangerous states and safe states. At the same time, hard failure modes are desirable on the paths from safe to hazardous (but unavoidable) states. An unsafe state, where a hazardous output such as a command to launch a weapon can be produced, should have at least one, and possibly several, hard failure modes for the production of the output command: No input received from the proper authority, no weapons launch.

- *Multiple paths should be provided for state changes that maintain or enhance safety. Multiple inputs or triggers should be required for paths from safe to hazardous states.*

In general, operators should be provided with multiple logical ways to issue the commands needed to maintain the safety of the system so that a single hardware failure cannot prevent the operator from taking action to avoid a hazard. On the other hand, multiple interlocks and checks should be associated with potentially hazardous human actions—such as a requirement for two independent inputs or triggers before a potentially hazardous command is executed by the computer.

15.5 Constraint Analysis

In addition to satisfying general completeness criteria, the requirements must also be shown to include the identified, system-specific safety requirements and to be consistent with the identified software system safety constraints.

- *Transitions must satisfy software system safety requirements and constraints.*

In a system hazard analysis, hazards are traced to the software–system interface. Such hazards involve specific software behavior expressed in terms of the value and timing of outputs (or lack of outputs). In general, software-related hazards involve

- Failing to perform a required function: The function is never executed or no answer is produced.
- Performing an unintended (unrequired) function, getting the wrong answer, issuing the wrong control instruction, or doing the right thing but under inappropriate conditions (such as activating an actuator inadvertently, too early or too late, or failing to cease an operation at a prescribed time).
- Performing functions at the wrong time or in the wrong order (such as failing to ensure that two things happen at the same time, at different times, or in a particular order).
- Failing to recognize a hazardous condition requiring corrective action.
- Producing the wrong response to a hazardous condition.

Constraint analysis on the software requirements specification includes a reachability analysis to determine whether the software, as specified, could reach the identified hazardous states.

- *Reachable hazardous states should be eliminated or, if that is not possible (they are needed to achieve the goals of the system), their frequency and duration reduced.*

It is not always possible to enforce a requirement that the software cannot reach hazardous states—sometimes a hazardous state is unavoidable. But this possibility should be known so that steps can be taken to minimize the risk associated with the hazard, such as minimizing the exposure or adding system safeguards to protect the system against such states.

The type of analysis required to guarantee consistency between the software requirements specification and the safety constraints depends upon the type of constraints involved. The presence of constraints can potentially affect most of the criteria that have been described in this chapter. Some types of constraints can be ensured by the criteria already described; others require additional analysis. For example, basic reachability analysis can verify that only safe states are reachable.

Basic reachability analysis may need to be extended, however, to consider additional constraints on the sequence of events. To illustrate, consider a simple

control system to move the control rods in a reactor up and down. The output actions to move the rods may be properly reachable and the paths robust. In addition, a constraint may require that a rod not be allowed to move within 30 seconds of its previous movement. To guarantee this constraint, all transitions where a MOVE ROD1 command can be issued must first be identified. Path analysis can then be used to find the sequences of events that will make the software issue two consecutive MOVE ROD1 commands. By showing that all possible paths described by these sequences will take at least 30 seconds to traverse, the constraint is guaranteed to be satisfied. If all the criteria described in this chapter for complete specification of timing requirements are satisfied, this analysis should be theoretically possible for a state-machine specification.

More generally, the specification may be checked for a general *safety policy* that is defined for the particular system. This process is very similar to checking that a specification satisfies a particular security policy [180]. The following is an example of a general safety policy for which the specification could be checked:

1. *There must be no paths to unplanned hazardous states.*
   The computer never initiates a control action (output) that will move the process from a safe to an unplanned hazardous state.
2. *Every hazardous state must have a path to a safe state. All paths from the hazardous state must lead to safe states. Time in the hazardous state must be minimized, and contingency action may be necessary to reduce risk while in the hazardous state.*
   If the system gets into a hazardous state (by an unplanned transition that is not initiated by the computer such as component failures, human error, or environmental stress), then the computer controller will transform the hazardous state into a safe state (every path from a hazardous state leads to a safe state). The time in the hazardous state will be minimized to reduce the likelihood of an accident.
   There may be several possible safe states, depending on the type of hazard or on conditions in the environment. For example, the action to be taken if there is a failure in a flight-control system may depend on whether the aircraft is in level flight or is landing.
3. *If a safe state cannot be reached from a hazardous state, all paths from that state must lead to a minimum risk state. At least one such path must exist.*
   If a system gets into a hazardous state and there is no possible path to a safe state, then the computer will transform the state into one with the minimum risk possible given the hazard and the environmental conditions, and it will do so such that the system is in a hazardous state for the minimum amount of time possible.

It may not be possible to build a completely safe system—that is, to avoid all hazardous states or to get from every hazardous state to a safe state. In that event, the system must be redesigned or abandoned, or some risk must be accepted. This risk can be reduced by providing procedures to minimize the probability of the

hazardous state leading to an accident or to minimize the effects of an accident. For example, activation of a carbon dioxide firefighting system in what may be an occupied space may kill any occupants, but it may be necessary to prevent the loss of an entire ship. Such difficult decisions obviously must be considered and specified carefully.

15.6 Checking the Specification Against the Criteria

The actual procedures that can be used to analyze a particular requirements specification will depend on the form of that specification. The criteria for completeness of states, inputs and outputs, and the relationship between inputs and outputs are easily checked for any type of specification. Criteria for the transitions between states will be checkable to a greater or lesser extent depending on the formality of the specification, the size of the specification, and the availability of software tools to help with the checking.

Heimdahl has automated the checking of the robustness and nondeterminism criteria (Sections 15.4.4.1 and 15.4.4.2) for specifications written in RSML, and validated his tools on an avionics collision avoidance system [116, 117]. Additional tools are being created for safety analysis of RSML requirements specifications.

Some criteria can be enforced simply by using a specification language that incorporates enforcement in its syntax. For example, a language that requires specifying value and time intervals for all inputs and data age limits on all outputs will not require additional analysis. Even if the language syntax does not require specifying a particular characteristic of the inputs or outputs, a syntax that makes omissions immediately apparent will be helpful in locating them.

On many projects, requirements are not complete before software development begins. In addition, changes are often made as the design of the other parts of the system becomes more detailed and problems are found that necessitate changes in the desired software behavior. It is therefore unlikely that the analysis will be completed before software design begins. To avoid costly redesign and recoding, the requirements specification and analysis should be as complete as possible as early as possible. Realistically, however, some of the analysis may need to be put off or redone as the software and system development proceeds.