Chapter 12

Input/Output: Interfacing the Real World

Consider a (chemical, nuclear,...) process plant controlled by a computer system. The design of such a process plant includes global dependability analysis by process experts, for whom the plant is a system of which the computer control is a subsystem. This dependability analysis may range from cost-effectiveness considerations to safety assurance through hazard analysis. The dependability models and mechanisms presented in other chapters describe how such a subsystem may contain and recover from its own misbehaviour.

The connection between the computer system and the plant has significance as a subsystem in its own right. The issue of interest for this chapter is to consider means whereby the dependability of this subsystem may be commensurate with, and contribute to, the dependability properties of the whole plant. At some stage, the process experts must consider whether the dependability properties required of the plant itself permit a design in which, for instance, a single piece of wire emerging from the dependable computer control subsystem is the only output controlling a crucial valve or pump or heating element. There are many obvious risks associated with such a design, which need not be enumerated here.

If such risks are unacceptable, the engineers concerned may require the dependability model to extend its boundaries beyond those of the control subsystem into the mechanics of the plant itself. This could entail, perhaps, such well-established "mechanical voting" techniques as on/off valves or other equipment being duplicated or tripled or even quadrupled; a common example is a series/parallel arrangement of valves and pipes (see figure 1).

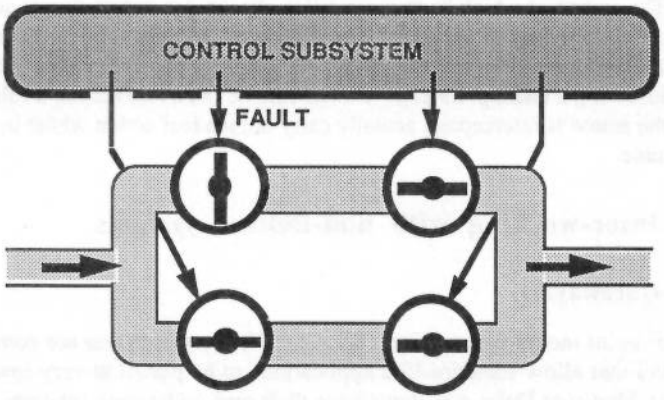


Fig. 1 - Example of "Mechanical" Voting

We are therefore concerned with input/output operations that cross the boundary of a Delta-4 system and interact with, or observe, entities in the enclosing environment that are not subject to Delta-4 models or assumptions. These entities may be complex and based on computers (commonly called "intelligent"); an elaborate example and special case of which is that of an external networked system, which is discussed in section 12.1 below. Alternatively, these entities may be primitive electromechanical devices (commonly called "dumb"), as in the above example; the rest of the chapter is concerned with various examples of this case. While the subject falls outside the remit of the Delta-4 project itself, it is worthy of discussion at length, since any system based on Delta-4 must find solutions and there are indeed opportunities inherent in the architecture that should be identified.

In this chapter, semi-active replication (see section 6.7) is revisited many times; the model seems to be well-suited to solving many problems of interaction between Delta-4 and the outside world. Two major distinctions between the Delta-4 models of semi-active and active replication are of particular importance in this chapter and so are summarised here:

- Active replication does not require a host to exhibit the fail-silence property (indeed, the discussion is made under the assumption that the host is not fail-silent) whereas semi-active replication, like passive replication, does require the host to be fail-silent.
- Whereas active replicas have a symmetrical relationship and during execution there is no assurance as to which replica will perform some computational step first, semi-active replicas are like passive replicas in having an asymmetry in their relationship both in terms of authority (which replica is "in charge" of computational progress and decision making) and time.

There are also two major distinctions between the Delta-4 models of semi-active and passive replication of particular importance in this chapter; again, these are summarised here:

- In passive replication, the computational overhead of taking, and the message size of transmitting, checkpoints is such that there is good reason to limit their rate. This is reflected in the discussion below, where a complex of output activity is assumed to occur between two checkpoints. In semi-active replication, the construct which "corresponds" to checkpoint taking and sending is action notification, which is cheap in both computation and communication terms; consequently it is appropriate to associate such notification with each individual input or output action.
- The construct which "corresponds" to checkpoint installation is actual execution of code, including in particular, code intended to cause (and in the leader, having just caused) a real input or output action. Unlike a passive replica whose new state on installing a checkpoint is as-if it has carried out a real action, a follower will, unless the action is intercepted, actually carry out the real action whilst in transit to the new state.

12.1. Inter-working with non-Delta-4 systems

12.1.1. Gateways

The MCS bi-point model (see section 8.1.4.1) offers services that are compatible with ISO standards and that allow standard ISO applications to be ported at very low cost in an MCS environment. However Delta-4 systems have their own addressing schemes up to the highest level of the OSI model: the inter-working with non-Delta-4 applications therefore needs a gateway function that must be performed within the application layer. Inter-working is illustrated in the following figures. Figure 2 shows inter-working between applications running

in a Delta-4 environment and a separate ISO environment. Figure 3 illustrates the point that this inter-working may take place on the same LAN because of the compatibility with standards that has been maintained at the low layers of MCS.

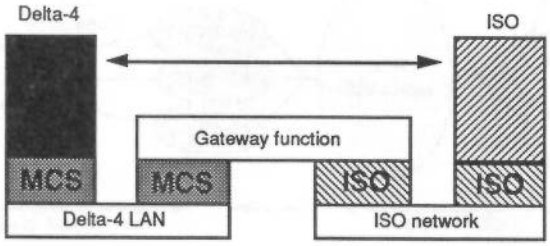


Fig. 2 - Inter-working between Applications running in Delta-4 Environment and ISO Environment

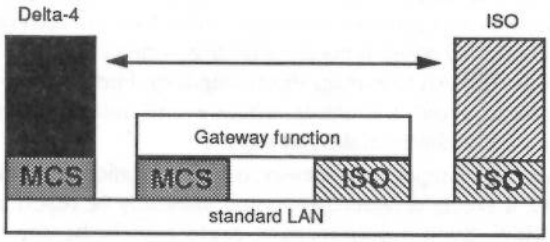


Fig. 3 - Co-existence and Inter-working of Delta-4 Environment and ISO Environment on a Single LAN

A gateway is thus an application-layer relay that provides a migration path for inter-working with other network architectures. Such a gateway interfaces both the Delta-4 world and the outside world. However, dependability cannot be conferred on this interface through use of Delta-4 techniques alone to replicate the gateway function on several nodes. Although these techniques offer transparent redundancy of the gateway function within the Delta-4 system, such transparency is not obtained for the non-Delta-4 applications outside the Delta-4 system. Although, later in this chapter, proposals are discussed whereby Delta-4 can contribute to the provision of dependable interfacing and the replication of primitive sensors or actuators, the provision of fault-tolerance to an ISO based application is quite another matter. This section will therefore not discuss how fault-tolerance might be achieved in the non-Delta-4 world, but will instead examine how Delta-4 techniques might help to preserve the availability of the interface between the two worlds. With this in mind, we now examine two ways of implementing such a gateway, based first on the active replica technique and second on the semi-active replica technique.

12.1.2. Replicated Gateway with Active Replication

In this approach, the gateway function is actively replicated with respect to its interactions within Delta-4. Each time the gateway is used to interwork with a non-Delta-4 (e.g., ISO) Application Entity (AE), each replica of the gateway function must establish an ISO connection to this AE. The ISO AE must explicitly manage the consequences to itself of this redundancy. It will receive several copies of each message from the Delta-4 world, and must separately send

copies of messages destined for the Delta-4 world on several links. An example of such inter-working is shown in figure 4:

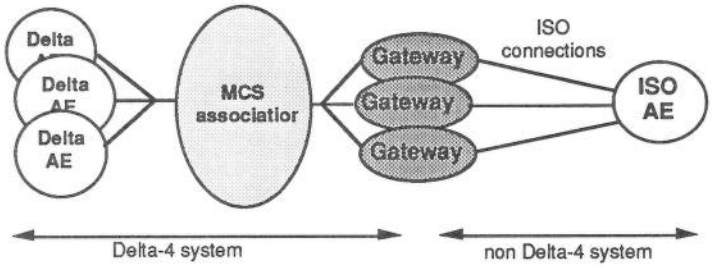


Fig. 4 - Gateway with Active Replication

In the figure, a replicated Delta-4 Application Entity inter-works with an ISO AE through a replicated gateway. In the Delta-4 system, MCS hides the degrees of replication of the replicated AE and the replicated gateway. In the non-Delta-4 system each replica of the gateway function must establish an ISO point-to-point connection with the ISO AE. Some software must therefore be added to the ISO AE to manage this multiplicity. Furthermore, the gateway must be carefully programmed to avoid a situation where events related to the ISO connections compromise the replica determinism of the gateway.

On the Delta-4 side of the replicated gateway, output validation will be carried out by Delta-4 voting mechanisms; a voting disagreement would normally be reported to Delta-4 System Administration by MCS. Such a disagreement could equally be due to a fault occurring somewhere on the ISO world; either in the communication on one of the ISO connections, or in the host environment where the ISO AE executes. The consequence is that the Delta-4 voting mechanism in the gateway can help to discover some misbehaviour outside the Delta-4 world. This must be balanced against the fact that fault treatment (diagnosis and repair) can become very complex, especially where no proper error containment domains have been constructed in the non-Delta-4 world. Whilst it is possible to arrange for the Delta-4 error-detection mechanisms to provide input to suitable fault-management applications that exist or could be created in the non-Delta-4 world, this approach should be seen as offering improved availability of the gateway rather than error detection.

Reconfiguration of the Delta-4 side is automatically performed in the Delta-4 system. On the non-Delta-4 side, disconnection of the failed replica and eventual reconnection of a new replica must be explicitly programmed. When a new replica of the gateway is installed, some actions must be performed on reconnection on both sides of the ISO connections to synchronize the gateway replicas with respect to ISO communication. In short, this solution implies that much of the machinery transparently provided by MCS in the Delta-4 system must be explicitly reproduced above ISO point-to-point communication in the non-Delta-4 system. Moreover, this management must be added to any ISO AE that is intended to inter-work with the Delta-4 system. This approach is therefore rather expensive and should be reserved for a few particular cases that require a high availability and fast recovery of the gateway function.

12.1.3. Replicated Gateway with Semi-Active Replication

The following approach presents a lower cost in terms of functionality having to be added to non-Delta-4 components. The model used will be recognised as corresponding to that of semi-

active, or leader-follower, replication (see the introduction above). Although support machinery for this model has been developed within Delta-4, the description given here is of what is necessarily an application-level implementation.

At any one time, only one gateway replica establishes a connection to the ISO AE, as shown in figure 5:

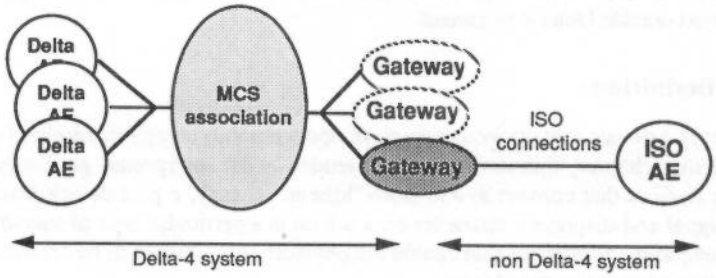


Fig. 5 - Gateway with Semi-Active Replication

All gateway replicas are programmed so as to appear to be true replicas when viewed from the Delta-4 system. However, they do not behave identically with respect to communication with non-Delta-4 AEs; in this direction, only one replica is seen as "active". Although all replicas receive the Delta-4 messages to be forwarded to the ISO AEs, only the "ISO-active" (leader) replica forwards them. The other replicas (the followers) instead queue them temporarily; when the leader receives confirmation of delivery from the destination ISO AE, it sends a checkpoint that allows the followers to delete the corresponding message from their queues. In the other direction, only the leader sends the messages it receives from an ISO AE by way of MCS. Such messages can be sent by using the Leader-Follower support of MCS, which allows the leader to send in the same atomic communication a message to a destination together with a checkpoint to the followers. If the leader gateway replica fails, all replicas are informed and a new gateway replica will take the role.

From the ISO AE's point of view, this solution does not totally hide the redundancy of the gateway. Some management of reconnection is required when the "active" gateway replica fails, with controls to ensure that messages are not lost or duplicated on the ISO connections when disconnection and reconnection occur. The Leader-Follower replication technique requires a fail-silence assumption on the hosts on which the gateway function resides. In the case of a gateway to an external world, this assumption is implicitly extended to this whole world. The disadvantage of this model is therefore that it does not provide any help for the detection of errors coming from the outside world; in the absence of application-level safeguards, such errors may propagate into the Delta-4 system.

The advantage of this approach is that it offers improved availability of the gateway function, with a minimum of essential additions on the non-Delta-4 side. It is therefore the recommended way to implement gateways between Delta-4 and non-Delta-4 systems if the risk of error propagation mentioned above is considered acceptable.

12.2. Inter-working with the Physical Environment

We will now turn attention to "dumb" peripherals, which are the normally understood domain of Input-Output. Several of the issues of the previous section are related to or recognisable in

what follows. Moreover, some of the approaches identified below appear general enough to apply equally to communications systems, microprocessor-controlled peripherals and the like.

There are two major divisions to the discussion, for which the issues involved are very different. The first, for which Delta-4 is able to offer significant support, is concerned with the acquisition of information from the environment; the so-called "sensor" problem dealt with under section 12.2.2. The second is the converse, the so-called "actuator" problem discussed under section 12.2.3; a more difficult issue, in which particular cases are straightforward but explicit support outside Delta-4 is needed.

12.2.1. Definitions

A transducer is a device that converts energy in one form into energy in another form. For the purposes of this chapter, this definition is intended to be interpreted generally and is not restricted to devices that convert to and from "kinetic" energy; e.g., a device that receives an electronic signal and displays a character on a screen is a particular type of transducer. To the necessary generality, all devices that enable the physical environment to be "read" or "written" by a computing system are transducers.

However, the terms input transducer and output transducer are used in this chapter to refer to devices (normally electronic) which are connected (e.g., by cabling) to the external environment but are physically inseparable from and so must be considered part of the host: examples are analogue-to-digital converters and output driver transistors. The devices in the external environment to which such transducers are connected, whilst themselves transducers according to the above definition, are distinguished respectively as sensors in the input case and actuators in the output case.

12.2.2. Input Sensors

Sensors (of temperature, flow, pressure, torsion, weight, level, position, direction, torque, etc.) are in many cases notoriously unreliable devices. To perform dependable computation on input from a basically unreliable sensor is to risk encountering the "garbage-in, garbage-out" phenomenon. Consequently, mechanisms for achieving and managing sensor and access mechanism redundancy, to ensure consistency and tolerate faults, are important. Below, we will evolve models that might be appropriate and coexist naturally with a Delta-4 system.

Three cases should be distinguished:

- A single sensor accessible from a single host (and therefore in the same fault containment domain as the host),
- A single sensor accessible from several hosts (via separate paths as shown in figures 6 and 7) (sensors and hosts are in different fault containment domains),
- Multiple (redundant) sensors each attached to different hosts.

12.2.2.1. Non-Redundant Sensors Accessible from a Single Host. The access mechanism is to perform non-replicated parameter value acquisition (including conversion, and other preprocessing if convenient). At the end of a (hopefully small) non-replicated chain of hardware and software components outside the domain of Delta-4, a single software component in effect "represents" the sensor within the Delta-4 system. There may indeed only be one identifiable acquisition component, though this is unlikely if, for example, Fieldbus-based I/O equipment is used. This chain of acquisition components up to and including the Delta-4 component is therefore collectively referred to in this chapter as the representative of the sensor.

Note the similarity between a representative and a "transformer" as described in chapter 7; it may indeed be convenient to implement such representatives as Deltase transformers.

The structure of a single representative accessing a single sensor can be considered as a logical subsystem. The representative is the only source of sensor values to the rest of the system. It can nevertheless interact with true replicated components, the characteristic of such interaction being assured by Delta-4 mechanisms to have the required property that all replicas receive identical values at identical points in computation. As an example, consider replicated Deltase client objects making an RPC to a non-replicated sensor-representing server/transformer.

If a host fails, the sensors attached to that host are lost. We now explore dependability first through representative redundancy and second through sensor redundancy.

12.2.2.2. Non-Redundant Sensors Accessible from Several Hosts. The next refinement is to consider the case where the reliability of a single sensor is considered sufficient, but the dependability of a single representative is inadequate (or thought to be a weak-link in the "dependability chain"). Can the representative be replicated?

To achieve representative redundancy, it is necessary for the sensor to be physically connected to a suitable input transducer on each station on which replicas of the representative might reside, i.e., the relevant replication domain. A point of concern is whether there are input transducer failure modes that are common-mode (for example, can an analogue or digital input transducer short-circuit the input wire). However, unlike the complementary case of outputs which is discussed below, there is little evidence that this is a serious problem.

In section 12.2.2.1, it is pointed out that within a Delta-4 system, replicas are assured by Delta-4 mechanisms to acquire identical values from a representative. However, interaction with the external world is outside the scope of Delta-4 mechanisms; replicated representatives under active or semi-active models of replication have no such automatic assurance of receiving exactly the same data when reading an input value from a sensor. If each of a set of active or semi-active replica representatives is allowed to interrogate an external sensor independently and directly, it will do so at a slightly different time. Even if perfect synchronism could be assured, it is unlikely that two replicas will receive bit-for-bit identity in a parameter value provided by the configuration given in figure 6, which shows normal analogue input preprocessing (analogue-to-digital conversion followed by linearization and normalization, though the latter two processes are normally deterministic and so do not compound the argument).

Even in the simple case where replica determinism is assured by computational-path-identity (see section 6.4.1), active replication of *representatives* is generally not possible. However, we now show that it is straightforward under the passive-replication model, and is possible with support from the application support environment under the semi-active replication model (see figure 7). Note that although figure 7 shows a similar physical configuration to figure 6, the replication models concerned are based on fail-silent nodes.

Under the passive-replication model, checkpoints are taken each time the sensor value is provided to other computation. Under the semi-active replication model, a somewhat different approach must be adopted needing environment support. When a sensor value is requested by a replica, this request must be treated differently depending whether the replica is a leader or follower. Only the leader actually collects values from a sensor. This value is then immediately sent by the support environment to the followers, as described in chapter 9 for propagating a consistent opinion on time. As in that case, the local environment then supplies the propagated value when a follower requests the same reading.

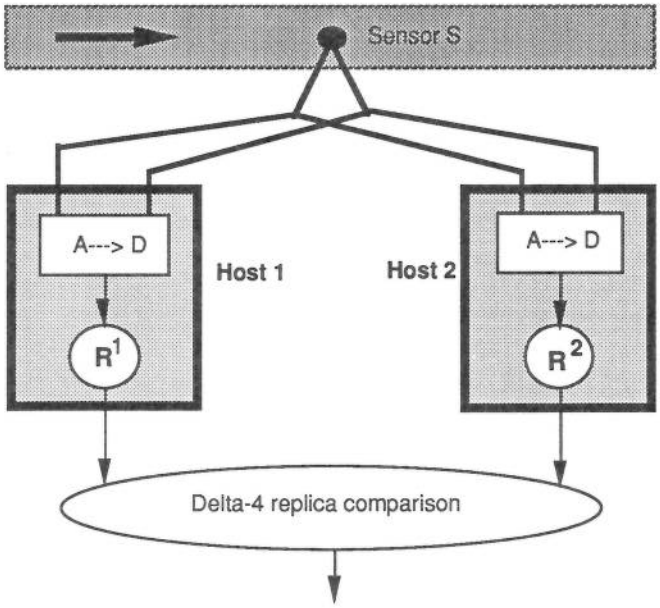


Fig. 6 - Single Sensor read by Two Active Replicas

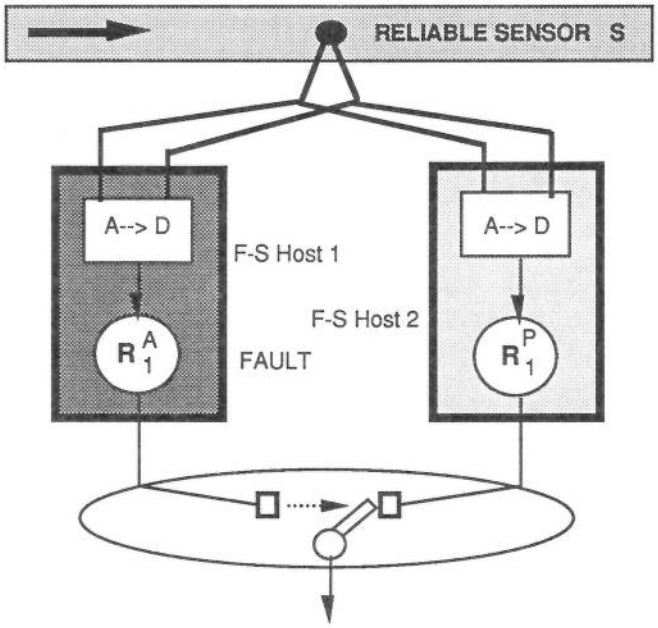


Fig. 7 - Single Sensor read by Two Passive or Semi-Active Replicas

By this means, the principles of semi-active replication are met; exactly the same reading is provided to each replica, even though the internally-generated requests are separated in time by a delay up to the maximum latency between the leader and its slowest follower. In the case of a failure of a leader after a sensor reading has been made but before this reading has been sent to the followers, the reading has not yet influenced the computational progress of the object. Providing the sensor is, as we are here assuming, a "pure" input device, the reading can safely be discarded and another one taken by the new leader replica.

Section 7.2.1 discusses various Delta-4 transparency mechanisms and the means by which these are managed. The model described here is one in which at least part of the distinct behaviour required of a leader replica and a follower replica could conceivably be provided, with assistance from libraries and tools, in a programmer-transparent way. Generic support mechanisms require that suitable abstractions be identified which nevertheless permit the many different actuator interfaces which might be encountered to be constructed. A less ambitious approach is to construct support libraries, language mappings and tools for each interface of interest.

A point of concern is that both passive and semi-active replication require a station to be fail-silent and the above discussion assumes that this property extends to the input transducers and all other components of the representative. This raises the design issue as to how to construct, for example, fail-silent analogue-to-digital converters.[1]

12.2.2.3. Sensor Redundancy. Consider now a set of sensors each intended to measure the same plant parameter (the flow in a pipe, the direction of motion,...) and each of which is connected to a (different) single station on which a single representative executes.[2]

The redundant sensor model that develops naturally from the above starting point is termed the "Rivals" model, in recognition of the fact that different sensors intended to measure the same parameter are not, and cannot be modelled as, true replicas. Even when correctly working, they will not provide bit-for-bit identical results other than by coincidence; this may or may not happen often, but cannot be guaranteed. On the contrary, they should only be thought of as providing similar results, where the degree of similarity is sensor-representation-specific[3] and subject to the laws of physics and perhaps statistical characterization. Nevertheless, to permit proper design, a formal definition of the (highly application-specific) meaning of this "similarity" must be captured in an application requirements specification.[4]

Judging what is to be considered a consensus value in the presence of sensor faults for the parameter concerned is necessarily application-specific (indeed, derivation of consensus even in

---

1 In this particular case, a design approach may be based on the so-called "inverse" method: a D/A converter is used to convert the digital value back to an analogue value that is then compared with the input value by an analogue comparator (with appropriate hysteresis). However, the analogue comparator (the need for which arises because of the approach taken; it is not an "inherently required component" of the conversion function) must be designed so that its failure leads to silence. The requirement has been changed and perhaps simplified, but not solved. It can be solved; the point is, however, that the design issues involved in any such exercise must be fully confronted.

2 More complex possibilities, where sensors are redundant and each sensor has replicated representatives, are ignored in this discussion for simplicity. In the event that such a design approach proves to be justified, the principles described here may "simply" be combined.

3 There may not even be type-consistency: the model given here can cope with, for example, one sensor representation of "direction" as integer degrees [0..359], another as floating-point radians [$-\pi..+\pi$], and a third as an unnormalized fixed-point Cartesian coordinate *(X,Y)*

4 The fact that identity is not sought between rival sensor representatives permits a spectrum of possibilities. There is not only no need for sensors to be of identical type but, in fact, the representatives might work in very different ways. For example, one "opinion" on the flow rate in some pipe might be derived directly by means of a flow sensor, and a rival "opinion" might be derived indirectly by calculation of the rate of change of level in a destination vessel.

the absence of sensor faults is highly application-specific) and a suitable algorithm should also be captured in the application requirements specification.[5] Well-known examples are: calculate mean, find median, discard extremes, etc., often these are used in combination and iteratively.

The provision of such application-specific algorithms falls outside the domain of standard support environment mechanisms; thus, in addition to the representatives, an application component that provides the consensus algorithm is required. The crucial observation that we wish to make here to complete this description of the basic model is that *the consensus component may be replicated* (see figure 8). The replication model is not constrained; active, semi-active, or passive consensus components are all possible.

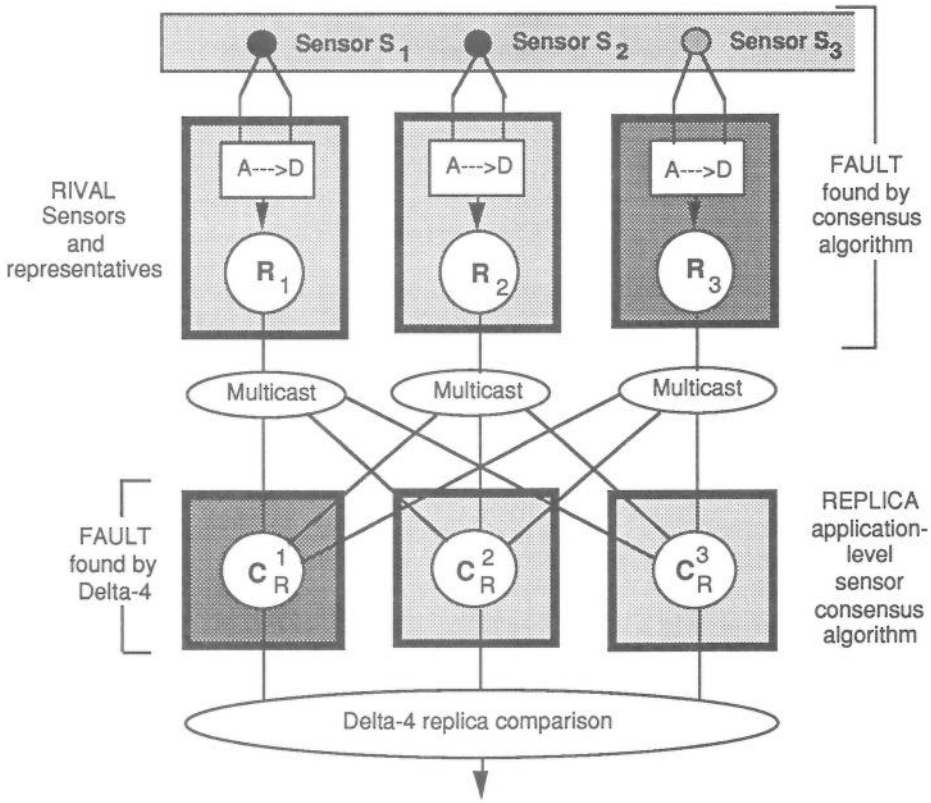


Fig. 8 - The "Rivals" Configuration of Multiple Sensors and Representatives

Note that, unlike replicas where failure is a managed event, detected and reported to system administration by MCS voting algorithms, representative/sensor combinations are found to have failed by the consensus algorithm. In order for such a discovery of failure to be treated as a managed event, application level consensus code must invoke, and so needs an interface to, system administration (see sections 8.2 and 9.5).

There are various means by which the apparent performance limitations of the model may be overcome, should they prove to be significant for some application. First, if a parameter is

5 The specification must, of course, also define the type of the result!

needed only in one replicated computation, there is no need for the consensus algorithm to be implemented in a separate address-space (component) to that performing that computation (Deltase offers a good way of still maintaining structural separation of function, described in chapter 7). The same might apply to the calculation of an indirect "opinion", as in the flow example above. Second, there may be no need for the sensor representative to be constructed to only acquire, convert, linearize and normalize a single value when prompted to do so by an RPC. Such components may instead use local cyclic activity or respond to externally generated events, in consequence maintaining values ready-transformed and date-stamped, for immediate response to RPC. Third, it is probable that a single component will represent a multiplicity of sensors and may indeed be constructed with a database-like interface capable of more complex local processing.

12.2.3. Outputs to Actuators

12.2.3.1. Interfacing Issues. We first consider problems associated with the design of output-transducer-to-actuator interfaces and the nature of the output transducers and the actuators themselves.

As with inputs, Delta-4 voting mechanisms play no part in assuring the correctness of outputs, even though these are generated by replicas. If the host concerned is fail-uncontrolled, the output that appears at its interface may be incorrect. Even if the host concerned is in all other respects fail-silent, it is difficult even to define a meaning for this property with respect to the type of output signal with which this chapter is primarily concerned. For example, a digital output might have two active states, driving current and not driving current. Neither state corresponds to "silence"; indeed there is no obvious state that can generally fulfil this role, although mechanisms which isolate such outputs may be effective in some cases. For a standard 4-20 mA analogue output to a valve position actuator, there is again no condition of the output that is in any satisfactory sense "silent". Preserving the last correct output value may also meet the requirements of some cases but is not generally acceptable.[6]

Application code does not act upon the external environment. Instead, the action is carried out through the intermediary of special hardware components that we refer to as *output transducers* and perhaps through support environment software components that we refer to as *output transducer drivers*. Such intermediary components are conventionally designed to offer the programmer a useful procedural abstraction of the output action concerned. Assuming there are no design errors, when a procedure is called, an effect occurs at the interface between the host and the external world that can be directly interpreted as corresponding to the intended external result.

Where greater dependability is required than is provided by the single actuator representative, there are three cases of connection of replicated representatives to actuators to consider.

- "Simple" actuators, with no special provision for the connection of multiple output transducers.
- "Compound" configurations of "simple" actuators.
- Actuators with multiple interfaces.

---

6 Some valve position actuators have a mechanism to ignore outputs which demand a change exceeding some percentage of the total output travel. For these, "silence" could be approximated by issuing a demand to go to one limit; the output is ignored unless the actuator is already close to this position. However, this interpretation is with respect to actuator behaviour on transducer failure. In contrast, replicate transducer takeover on transducer failure might require "silence" to be an open-circuit. Such a condition may not be satisfactory to the actuator during the interim period, or as a safe state if there is no more redundancy. Actuator silence is not (generally) transducer silence.

The first case is discussed in the following section 12.2.3.1.1, where the general concept of actuator representatives is also introduced. The other two cases have sufficient commonality to be discussed together in the subsequent section 12.2.3.1.4.

12.2.3.1.1. Actuator "Representatives". In the highly specialized world of industrial actuator technology, it is rarely the case that an actuator is designed with replication in mind. For the task concerned, no special provision has been made for the connection of more than one output transducer. The actuator possesses, in effect, a single interface. The system must offer such "simple" actuators a single, validated version of the output in the form required, masking, if necessary, any time skew between different replicas.

If the dependability consequences are acceptable, a single synchronized version of the output may be achieved by constructing a single active intermediate software component driving a single output transducer. Validation is then performed by Delta-4 mechanisms on an internal-to-system representation of the output, e.g., a message specifying a new value. This intermediate object is, in effect, representing the actuator within the Delta-4 system; a dual of the situation described earlier for sensor representatives. This can be expected to be a common configuration and would typically be based upon a simple conventional Fieldbus link between a Delta-4 system and an "intelligent" peripheral.

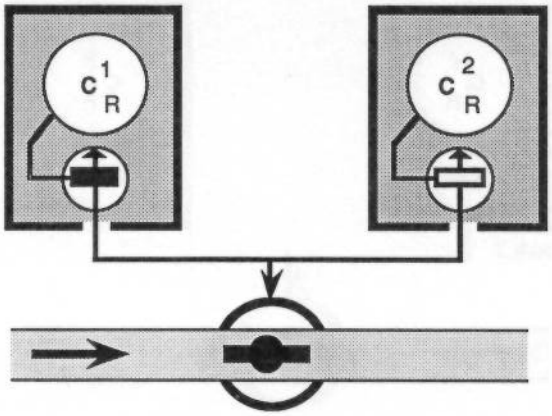
Where the reliability of the single actuator is considered sufficient, but the dependability of the single representative is not, it is necessary to physically connect multiple transducers to the actuator interface in such a way that *from the point of view of the actuator interface*, there is only one active transducer. See figure 9, which shows two examples of this. The first is a simple configuration of a pair of driver transistors each able to cause a simple valve actuator to go to the "on" position. The second is the very common traditional *changeover* system, where devices such as solid-state switches or even electromechanical relays are used as a basis for isolating an output transducer. Requirements are imposed on the replica driving of node output transducers, and upon the nature of the interconnections between the output transducers and the "simple" actuator.

12.2.3.1.2. Node Requirements. A single output at a time is naturally achieved using passive replication, but semi-active replication can also be given this property if, at the follower host interface, a call for output is prevented from having an effect. When an output procedure is called by application code on some host, an effect normally occurs at the host output transducer that corresponds to the intended external result. If a more elaborate output transducer driver is constructed such that, although all replicas are active in that they execute the output call, then the effect only occurs at one host transducer; at the other replicas' hosts, the transducer drivers "intercept" the call for action.

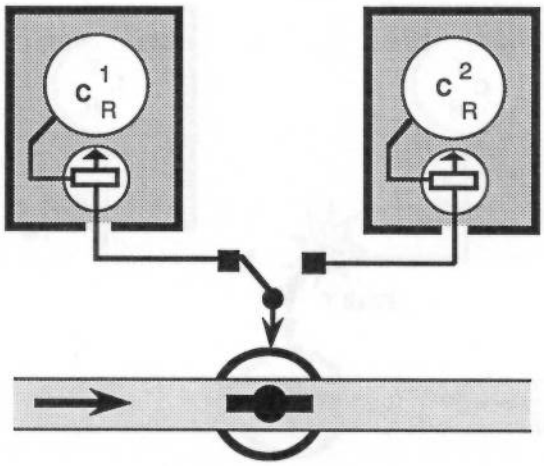
Such a mechanism introduces a bias into the status of the replica that is "really" performing the output, so naturally belongs with replication models where such a bias is already evident and managed. Semi-active replication has such a naturally-occurring bias.

In both passive and semi-active cases, output fault containment is required. Since, under these models, the node involved is fail-silent, it is tempting to appeal to the properties of fail-silence. When a node fails its output transducers are assured to be disabled in whatever way is appropriate for that class of output. As pointed out at the start of section 12.2.3, this is highly application- and transducer-specific.

12.2.3.1.3. Interconnection Requirements. The major issue in interconnecting multiple output transducers and "simple" actuators is that of common-mode failures.



An on/off valve driven by two output driver transistors



A simple mechanical output "changeover" system

Fig. 9 - Single Actuator Controllable from Multiple Sources

In the first example of figure 9, if the driver transistors are required to carry heavy current, they possess an unfortunate probability of failing into a short circuit condition, as a molten lump of silicon; the consequences are shown in the first example of figure 10). The valve is permanently "on" until the interface is disconnected.

In the second example of figure 9, if the basis for isolating an output transducer is inappropriate, the condition illustrated in the second example of figure 10 can arise. Although, in the configuration shown, the fault remains latent, there is no obvious means of detecting that the fault exists, the intended output transducer redundancy is not available, and there is a difficulty in replacing the faulty component on-line.

A fault in inter-station signal earthing can also be very difficult to detect. When a transducer fails, a current path for its newly-activated replica might not be through the intended route and so cause all sorts of problems, from excess noise-sensitivity to activating mains safety cutoffs.

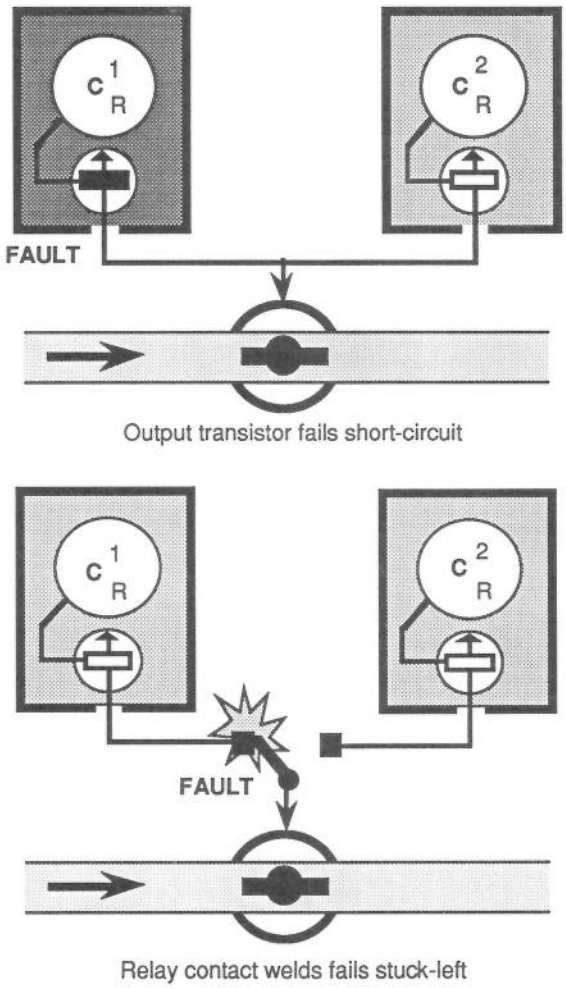


Fig. 10 - Common-Mode Failures

To summarise, the physical connections to the actuator from each of the nodes on which the intermediate voting object might reside must be made in such a way that there is no interactive interference (e.g., from output feedback sensors or unexpected current paths) and that no arbitrary failure of the output transducer on one node can prevent another output transducer from fulfilling the output requirement, within the necessary level of coverage.

Primitive output transducers rarely exhibit benign failure modes. Through feedback, the local fault-detection of each node concerned may be extended to the output transducer and perhaps beyond (the condition of the output is sensed and compared with the desired output). In the case of passive replication, the present primary replica, or in the case of semi-active replication, the present leader, or in the case of active replication, some nominated replica, is the master and hence connected to the peripheral device. Under as many conditions of failure as possible some form of isolation device should disconnect the output.

Due care must be paid to the peculiar properties of isolation devices. For relays, this includes such design issues as "wetting" current and contact material migration, the significant

chance of a contact welding shut, the very limited number and rate of operations, the long (relative to electronic switching) changeover times, open-before-close vs. close-before-open designs, vibration-sensitivity, etc. However, isolation thresholds are better than with solid-state switches, which can be important where cable runs are long, equipment is on different phases of mains or handles high voltages, earth-return-current design issues are troublesome and so on.

12.2.3.1.4. Voting Actuators. Where an actuator is explicitly intended to be used with replicated outputs, care is taken to provide for more than one interface, to isolate faults in these interfaces from each other, and to provide a solution to interactions between these interfaces, based on some model for how these interfaces are to be used and how and what failures are tolerated. Such actuators, and configurations of simpler actuators that exhibit similar properties, are now discussed. An example of a compound configuration of simple actuators, introduced at the start of this chapter, is the series/parallel combination of on-off valves.

Active replicas will each cause a version of some output to appear at the several respective host output transducers. We shall refer to these as "replica outputs". An actuator that is capable of accepting several loosely-synchronous outputs from such replicas and majority-voting on them locally to determine the action to be carried out, may use three or more of such replica outputs directly. Suppliers of computer control equipment may be required by the plant contractor to provide three or four replica outputs, to be voted externally by independently-supplied mechanisms of this variety.

Delta-4 provides an appropriate means to construct the active replicas and therefore the replica outputs (see figure 11), given that the degree of desynchronization experienced by active replicas can be assured to be within the bounds specified for the voting mechanism.

Where a difference in outputs occurs, one of the following conditions is the cause:

1) An unmanaged opportunity for application non-determinism
2) Excess desynchronism (beyond some specified limit)
3) Computational subsystem failure
4) Output transducer failure

The first two cases are, respectively, (1) an application design and (2) an application/support system interaction design problem.

The voting actuator is irrelevant in the presence of such problems, which are not covered by any replication model. The system is rendered unable to provide the required service by any such event. The voting actuator gives additional and valuable coverage against cases (3) and (4). It is therefore consistent with and supports the fail-uncontrolled host active replication model.

We now consider fail-silent host models. In the semi-active case, majority voting implies a leader and at least two followers. Outputs generated by the followers must not be intercepted, but must arrive at and be taken account of by the voting actuator. From the actuator point of view, follower failure and leader failure are equivalent. Indeed, the actuator "sees" no significant distinction between active and semi-active replication, other than that one output (that of the leader) will always precede the others.[7]

---

[7] In XPA, the leader precedes by at least a bounded amount (minimum leader-to-follower desynchronisation) and with a larger overall bound(maximum leader-to-follower desynchronisation). Since actuator voting requires the outputs from the leader and at least one follower to arrive, the time bound for the voted result is the difference between the two; this is more tightly bounded than are the replicates themselves.

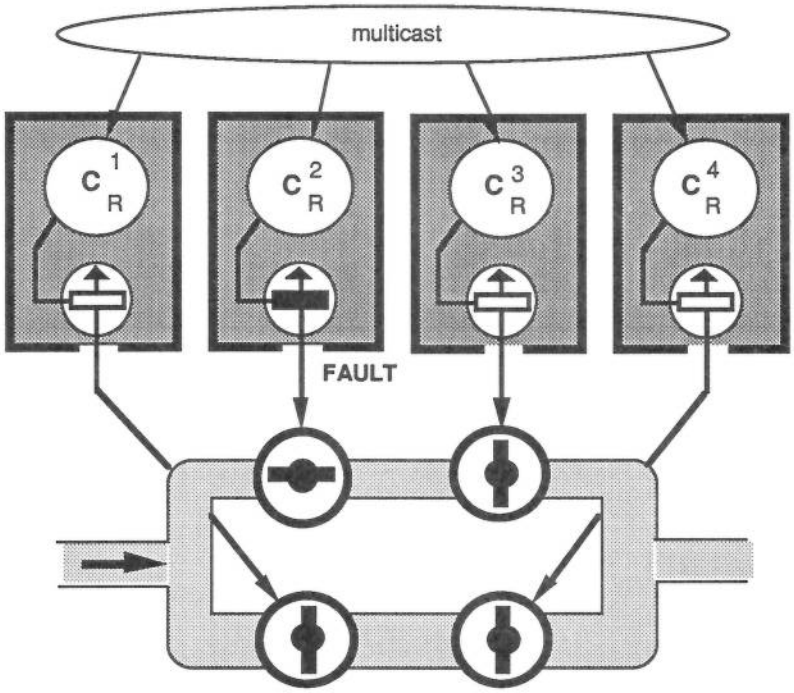


Fig. 11 - Active Replicas controlling a "Mechanical" Voting Actuator

A point made earlier in section 12.2.3 was the difficulty in extending the notion of "silence" to output transducers. However, with voting actuators, if this variety of host fault does not result in "silence", this can be tolerated. How far inside the host can this need for silence be relaxed in the presence of a voting actuator? Again, consider the possible causes of a difference in outputs. This might result from either (1) or (2) as above, for which the same conclusions apply in the semi-active case, or:

5) Computational subsystem lack of coverage (failure to exhibit fail-silence)

6) Output transducer lack of coverage (failure to exhibit fail-silence)

The voting actuator gives additional and valuable coverage against (5) and (6) which may otherwise be difficult to handle. The technique described in this section is therefore a means of tolerating fail-silent support environment lack of coverage in the particular case where this leads to output failure, and of relaxing the fail-silence requirement of semi-active replication with respect to the transducer hardware itself.

12.2.3.2. Output Error Detection and Recovery

12.2.3.2.1. Imperfect Output Error Containment. The discussion of "simple" actuators above assumes that fault-containment at an output transducer has been perfectly achieved. This may not be the case, e.g., an output transducer may not fail cleanly and a transient pulse might occur which causes the external world to transit to a different state to that represented in the control system, or it might not be known whether a particular output has or has not occurred at the point of failure. An approach to solving such problems is to use feedback or application knowledge to establish consistency between the control system and the

outside world, and then to resolve any difference between the discovered state and the desired state of the outside world. This approach, which also applies to faults arising with the actuators themselves, is discussed in the following sections.

Under passive replication, host hardware faults are tolerated by backward recovery to a previous valid state and subsequent re-execution (see chapter 6). This concept is difficult to extend to the real world, since restoring a previous state will not, in general, be possible. The real world can only roll forward; real actions cannot by any single mechanism or principle be rendered "undone". Once cash has been released into a cash terminal customer's hand, it cannot be taken back. Once a valve has opened and a chemical reaction has begun, it cannot be reversed. In the general case, devising a way whereby a future state of the global system is arranged to coincide with a previous state is perhaps impossible; instead, an acceptable future state must be reached.

For a cash terminal, the requirement might be to complete, if possible, the transaction correctly. Presumably cash should indeed be released to this customer if the transaction has proceeded to the point where it is mechanically ready for release. Can the system discover what amount has already been released? If this proves impossible, a secondary wish might be to limit the damage, perhaps in terms of financial loss or reputation.

Similarly, for the process plant example the requirement might be to complete the reaction correctly. The amount of reagent might be determined as a function of flow rate and time; what was the flow? When was the valve opened? Has a quantity limit or critical time been passed? If so, a secondary requirement might be to limit the damage, perhaps in terms of environment or plant down-time.

To deal with this type of situation, the notion of *forward error recovery* (see chapter 4) is used, and is discussed in the next section.

12.2.3.2.2. Computation Issues. With simple output actions, if no checkpoint is received, a real action may not have occurred at all, or may have been carried out by the primary replica which immediately failed prior to propagating the fact to the rest of the system. If the checkpoint is sent before the real action, this action might or might not be carried out by the primary replica, but the checkpoint information is that it will be. Rolling forward a computer control system that has experienced failure may therefore cause an output either in the first case to be repeated, or in the second case never to occur. Propagation of information in two physically disjoint directions (out to the real world, and internally to other computation) from a single sequential computation cannot be atomic.

With compound output actions, where several "primitive" real actions take place between checkpoints, the problem is one of dealing with the consequences of failures that occur while the compound action is actually being carried out. These consequences may be varied. The failure may have occurred immediately after an object took its first checkpoint, in which case the compound action will not yet have begun. It may have occurred immediately before the final checkpoint, in which case the compound action will be complete. Alternatively, it may have occurred somewhere in between these points, in which case the compound action may have been partially carried out.

Here, as in the cases examined in the previous section, the solution is to use *forward error recovery*. This involves executing application-level recovery mechanisms specific to the external system involved. The first requirement is to establish consistency between internal opinion of external state and actual external state; i.e., to achieve a "known" state. One way to achieve this is to correct the internal state, for instance by interrogating the environment to determine the actual situation: *has the cash been released? Has the valve been opened?* This requires that arrangements are made in the design — i.e., the necessary sensors are included — to obtain all relevant information. If complete interrogation is impractical, it may be

possible to make use of assumptions (e.g., the laws of Physics) which allow some properties of the environment to be inferred: *what is the temperature in the reaction vessel? How much reagent has been used?* Another, more interactive, possibility is to change the external environment so as to progressively reduce the uncertainty, i.e., carry out some initial compensatory actions that are guaranteed or can be reasonably assumed to place the environment and the system in a compatible state: *shut off reagent flow, flood reaction vessel with neutralizer*.

From the known state determined by such activity, the necessary actions must be applied to complete, or safely terminate and possibly restart the action anew (the nearest equivalent to the concept of "abort" in a roll-forward world): *dump neutralized reaction products for later analysis and recovery, scrub reaction vessel, restart with new reagents* (see figure 12).

12.2.3.2.3. At-Least-Once Semantics. The discussion so far has established that, in the general case of computer subsystems linked to non-dependable I/O subsystems, under host failure conditions:

- Where the semi-active or active/passive models are used, since output and notification/checkpoint cannot be atomically associated, a new leader or new primary replica cannot simply continue, but must pursue a forward error recovery procedure.
- Where the active model is used, there is no built-in mechanism to synchronize or bind input/output actions with computation, and, if the failed host is the one responsible for output, then an event must be raised to cause all replicas to pursue a forward error recovery procedure.

However, in the particular case of *idempotent* output actions, i.e., actions that do not cause incremental change and can therefore be repeated (for example, a valve position demand), application-specific forward error recovery procedures can be avoided if at-least-once semantics can be assured. In section 12.2.3.2.2 above, it is shown that if a checkpoint is sent before a real action, it is not known whether this action takes place or not. However, if, on takeover, a new primary performs the real action, then at-least-once semantics are achieved. Similarly, in the semi-active case, if a notification is sent before a real action is requested, then the new leader can achieve at-least-once semantics in the manner described below.

In the general discussion of forward error recovery above, the granularity of this recovery is assumed to be coarse. If at-least-once semantics are to be effective, the granularity must be made sufficiently fine for idempotence to be practical in the context of a dynamic controlled process. If the implementation involves an excessive inter-command latency as regards changing the valve setting, then the plant may not be controllable, or may move to some state that requires a full forward error recovery. Reducing the granularity ensures that the possible discrepancy between plant and control system is correspondingly reduced.

Semi-active replication allows short inter-notification latency because of the small network bandwidth consumed by such notification (see chapter 9). Thus, although the concepts discussed below can also be applied in principle to passive replication, such an approach would not be practical.

In what follows, a real output is assumed to be a primitive single-step or atomic transducer event at the leader's host. The last decision that is known to have been taken by a failed leader is defined in its last notification. We need to ensure that a new leader is not diverted to a different execution path prior to completing the real actions previously requested by the ex-leader.[8]

---

[8] A new leader can receive an external event and therefore be diverted to a different execution path at any subsequent decision point. As long as there are no externally visible consequences, it does not matter if the original leader had previously passed this point without diversion. Real-world actions are, however, such

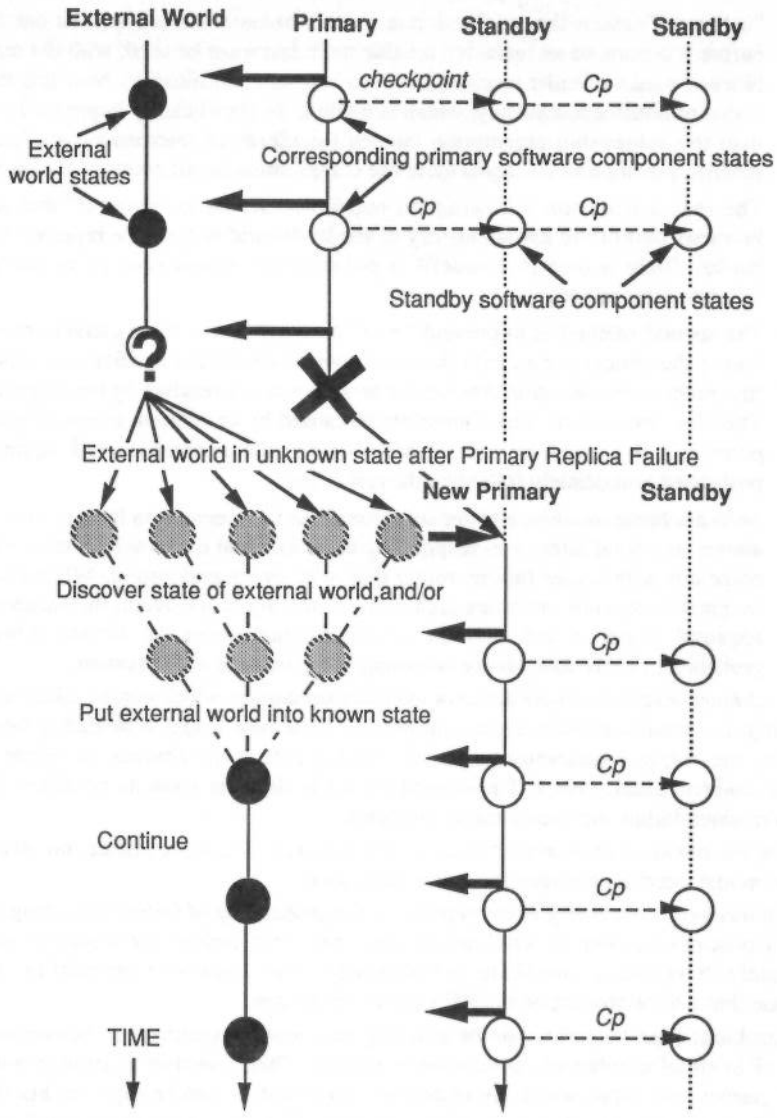


Fig. 12 - Forward Error Recovery

There are two methods by which this may be achieved. Both make use of the concept of a "notification point". This is a point at which, unlike a normal decision point, notification always occurs. A notification point may or may not also be a decision point.

- The first method is to locate such a notification point before the real action, with no decision points between this point and the real action. The real action can still be

---

externally visible consequences; to be assured that a different execution path does not "orphan" an action in this way, the new leader must not be allowed to divert execution at any decision points encountered between the last point notified by the previous leader and the point of action.

"orphaned" unless the notification is assured to have reached at least one follower before it occurs, so an inclusive reliable multicast must be used, with the real action blocked until the leader receives back this self-same notification. Note that this delay is that of notification latency, which is likely to be significantly larger (milliseconds) than the achievable preemption latency (hundreds of microseconds). Under this scheme, preemption latency is therefore compromised at all points of real action.

The first notification following the real action serves to "commit" that action; it becomes part of the known history of the leader and will not be repeated by a new leader. There is therefore benefit in positioning a notification point immediately following the real action.

- The second method is to prevent "new" external events from causing preemption during the critical period until the new leader emits its first notification, which is the first point in its execution that we can be sure was not reached by the original leader. This first notification cannot therefore be caused by an external event. A notification point has the necessary property of being self-originated, and again is best positioned immediately following the real action.

  Such a scheme involves a better compromise to the preemption latency time than the above; extended latency in responding to an external event occurs only when this coincides with leader failure, rather than with every real output. Whether either is acceptable depends on issues such as the criticality of the event, the hardness of the required response and, perhaps of significance under the second scheme, the probability of this coincidence occurring in the lifetime of the system.

Both schemes lead to a simple actuator interface semantics, which can be taken advantage of providing the actuator exhibits certain properties. If the new leader is treated as being so in all respects, the output is guaranteed to occur exactly-once in the absence of failure, and at-least-once (indeed usually-once if post-notification is done as soon as possible) for each occasion on which failure and hence takeover occurs.

That is, the *output is guaranteed to occur at-least-once*. If it occurs twice, no other output will occur on that interface between the two occurrences.

The chances of it occurring twice depend on the probability of failure occurring between action and post-notification. If NAC delays were tiny, this probability would be very low. Unfortunately, NAC delays are finite and adversely affect the above probability. This is a design issue that will be studied on the XPA prototype system.

The problem then becomes one of assuring real-world exactly-once behaviour in the presence of (a small number of) at-least-once outputs. This is simple to arrange with many actuators, particularly those which are primitive. "Open valve" can be repeated but the valve opens once; repeated demands are, in effect, ignored. This is a canonical example of a very general solution that can be applied to more complex actuators, given that there is some control over their design. This solution is to construct these as (mechanical, hydraulic, electromechanical, electronic,...) state machines, which, again in effect, ignore all but an expected subset of the universe of possible outputs when in each state; each acceptable output causes transit to a new state for which the same holds. The at-least-once variety of output repetition is tolerated by arranging to ignore any repetition of an immediately previous output; if repetition of action is required, it can be arranged by introducing intermediate outputs causing transit to specially-introduced intermediate states.

If the actuator is "given" but does not have such properties, there is a problem with this form of replication. Nevertheless, some quite complex cases are amenable to the approach outlined here. Consider the following "intelligent" actuator: A riveting robot has operations *step left, right, up, down, in, out,* etc., and also *fire, reload* rivet (it is immaterial here whether these

commands are pulses on individual control lines, simple codes sent over a serial or parallel interface, or something altogether more sophisticated). *Fire* is at-least-once in that it will not succeed unless *reload* has been issued since the previous *fire* command. The same is true of *reload* with respect to *fire*; these operations naturally alternate between a pair of states. *Step*ping is incremental, but since the steps are tiny and the position is sensed (as part of a closed control loop) before a *fire* command is issued, an occasional error of one additional *step* is naturally tolerated by the control philosophy.

12.3. Summary

This chapter has argued that extending the boundaries of dependability to encompass sensors and actuators must be managed in an application-specific way. Dependable input-output design is not amenable to generic support, although there are some cases where particular modules may be constructed to be consistent with a generic architecture; examples are indeed available from some manufacturers.

We do not consider it sensible to construct an extension to the Delta-4 environment to encompass the provision of dependable input-output. Much of the limited generic support that can sensibly be offered by an environment like Delta-4 is already present or planned. Where it is not (such as in the area of intercepting follower output calls), it will take experiment to establish the ideal mechanism and to create the appropriate tools. This work, whilst not inconsistent with Delta-4, has separate concerns to the central thrust of the present project, so should be pursued in another context, such as productization of the Delta-4 architecture.