There has long been a need for better definition of the audit and control aspects of data processing applications. This paper attempts to satisfy that need and thereby provide a framework for improving communication between systems analysts and computer scientists. It introduces the concept of spheres of control, *which are logical boundaries that exist in all data processing systems, whether manual or automated. The paper describes their essential properties and portrays them as they relate to each other in the batch, on-line, and in-line processing environments. Included are spheres of control that define process bounding for such purposes as recovery, auditing, process commitment, and algorithm (procedure) replacement.*

Data processing spheres of control

by C. T. Davies, Jr.

Not enough has been written about maintenance of integrity or recovery from incorrect processing discovered either during the processing or afterward. Similarly, not enough has been written about control over processing and the auditing of processing, or about the effects of processing in large, complex systems of multinode distributed data and multinode distributed processing. The general and typical case for multinode multiprocessing systems involves many humans and machines, each an active element in a network of active elements and each the holder of a subset or redundant copy of one or more data bases. Many transaction oriented business systems are examples of this multinode case.

The problems addressed in this paper are those of keeping processes from interfering with each other, returning a process to some previous and more acceptable state, preventing the use of created or updated information until it is known that the process will not have to be backed out, controlling processes in the same or different nodes, preserving order in a multiprogramming and multiprocessing environment, saving process results for subsequent audit to reduce the probability and significance of errors, and providing repeatability of process results as required by most auditable applications.

All applications have these problems, for which the user must find solutions. This paper describes solutions that are not application dependent and hence can be automated, resulting in a more error-tolerant and auditable sytem which all applications can use. In particular, these concepts, when implemented, allow management to better understand and control what is going on in a data processing system.

Any technique for controlling and auditing processing in a multinode multiprocessing system must first delineate the spheres of control, or logical boundaries, for each active element of the system. There are many different kinds of spheres of control, each maintaining a set of relations within a boundary associated with a particular kind of control. The boundaries of the spheres of control described in this paper are delineated by operators and descriptors. The kinds of control considered, for which there are potentially many instances, are process atomicity, process commitment, controlled dependence, resource allocation, in-process recovery, post-process recovery, system recovery, intraprocess auditing, interprocess auditing, and consistency.

Spheres of control exist for other purposes, but they are not considered here because of space limitations. Examples are privacy control, transaction control, and version control for both data and procedures (instruction data).

Process control

Process control ensures that the scope of processing at each level of operation in a hierarchy is defined by the set of operation codes implemented at the next lower level. At each level, only the operation codes of the next lower level are defined (made accessible). Each level is merely an implementation of the operation codes that invoked it. That is, each operation code may require many other primitive operation codes in its implementation. Each of the primitive codes in turn may be implemented by one or more different operation codes. The languages used may differ considerably from level to level.

process atomicity

A call to a subroutine is an example of a primitive at one level of implementation invoking a set of primitives at a lower level. The processing of an operation code at a given level at a given instant is called an atomic process. Process atomicity is determined by the amount of processing that one wishes to consider as having identity. It recognizes that data processing involves discrete units of process which we call digital. Therefore, regardless of the implementation, an atomic process is performed either in its entirety or not at all. It may be a payroll application to some people, the square root subroutine to others, or perhaps a machine instruction like ADD, SUBTRACT, or MOVE, or a microcode instruction.

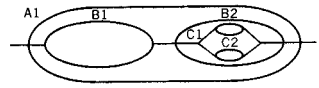
Process control also ensures that the information required by an atomic process is not modified by others, and it limits the extent to which another process can depend upon the updates made by this process, as described below.

Atomic processes provide the greatest possible implementation independence. If a function is rewritten, only the changed function and the lower layers in the hierarchy are affected. Frequently, also, it is desirable to have one process terminate and another start at the interface between two active elements, particularly if they are geographically separated. Process subdividing allows for a defined interface that provides compatibility independently of the specific implementation of either atomic process.

The atomic process, or action, as it is often called in commercial data processing, bounds the unit of function to be executed and the interpreter to be used. For example, one atomic process may be described in COBOL and another in PL/I. A function may even be written in one language and call a subroutine written in a different language.

In passing, it is interesting to note that in a system of many nodes, each having a potentially different interpreter (instruction set) and with each procedure perhaps written in a different language, it is mandatory to know the interpreters needed, their availability, and their location. Otherwise, it is likely that, for example, a PL/I sequence of instructions may be sent for execution to a node that understands only APL. It is not necessary that the interface between them know the implementation of the other nodes or levels, other than in a semantic sense as viewed through a common, defined interface.

Figure 1 Spheres of control—process atomicity



Each sphere of control is an atomic process (single operation) when viewed from the next higher level of control.

In summary, process atomicity is the control over processing that permits an operator to be atomic at one level of control, while the implementation of that operator may consist of many parallel or serial atomic operators at the next lower level of control. Thus there is a fully nested structure which provides implementation independence at each level relative to lower levels.[1] Every defined atomic operator can be given a name. Only an atomic operator has the potential to have versions, and only defined atomic operators can be moved to and executed in other nodes.

Figure 1 illustrates a hierarchy, or nest, of atomic processes. A1 is a function as viewed from outside A1. B1 and B2 are what A1 sees, and C1 and C2 are parallel processes visible to B2.

process commitment

While a function is in process, changes of state of significance to the function are being made only by that function or are expressly known to and permitted by that function. In a payroll application,

for example, the first job accepts time cards and writes paychecks, a transaction history, and deductions, some of which may provide input to an employee stock purchase program (a second job). From a logical point of view, one could run only the time-card input and paycheck output. To reduce the possibility of error, however, it is desirable to hold the paychecks until after the transaction history has been audited and the stock purchase program run.

In case a rerun is needed, holding the paychecks establishes a single checkpoint which is independent of the error detected and the number of functions processed since that point. In particular, holding the paychecks prevents other functions from depending upon them for process commitment, so the writer of the paychecks can revoke them without having to be concerned about unfavorable consequences.

Preventing process commitment by holding (controlling) the use of its results permits the system to perform a unilateral backout (process reversal or undoing) over much larger units of processing. *Unilateral* here means without having to request permission from each participant in a process. Thus mistakes not detected until late in the processing can be corrected, although response time will be degraded.

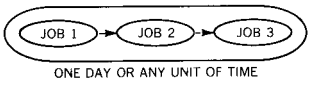
So far, the containment of process commitment has been discussed as though it were preplanned. That is not always the case, however. When an error is detected and its source not yet fully determined, or the action required to correct it is not fully known, the effects of processing should be contained until a determination can be made as to whether releasing the effects will adversely affect subsequent processing. Constraining a potentially erroneous process so that output and updates are not released until the processing no longer need be able to rerun to a potentially different conclusion is called dynamic control over commitment. This boundary of control can extend to as much processing as is economical to control.

The sphere of dynamic control is extended over process commitment and permits processing to continue rather than having to end abruptly. At best, when an error is better understood or is discovered to be nonexistent, the boundary simply can be removed so that the locked resources are unlocked. Thus a great deal of time will have been saved. At worst, a lot of processing will have been done needlessly. However, the time used will have passed, regardless.

In summary, process commitment control is the containment of the effects of a process, even beyond the end of the process. It can be used, at the expense of response time, as a relatively in-

expensive method of recovery. As an example, an entire day's processing may be contained within a sphere of control over commitment, so the entire day may be rerun to a different conclusion, but at the expense of not being able to commit to any of the results until the end of the day.

Figure 2 Sphere of control over process commitment



Boundary around the effects of a process, even after processing is complete, allows for independent audit and back-out and for rerunning to a different conclusion.
Preplanned for audit: static shape.
Suspected error: dynamic shape.

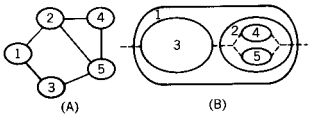
Figure 2 illustrates control over process commitment. Any data upon which any of the three contained functions (jobs 1, 2, 3) depends, or that has been created or modified by any of the functions, is assigned to the control sphere of all three until the sphere terminates. The effect is the creation of larger domains of process.

multinode process atomicity and commitment

Figure 3A shows a typical multinode network of active elements, with their associated data connected by communication lines. The nodes can contain either human or machine elements. Figure 3B represents one possible calling sequence for doing some work. The work is initiated in node 1, which calls upon node 3. When control is returned from 3 to 1, node 1 calls upon node 2. Unknown to node 1, node 2 calls upon nodes 4 and 5 to perform some work, and when the work is completed node 2 returns control to node 1. The key point is that completing work in any particular node does not necessarily mean that the results can be depended upon by, or committed to, some other process in the same node.

In particular, no output of any process can be depended upon by other than the next higher level in the processing nest regardless of its geographic location. In the event that the process encompasses more than one node, the other nodes have process agents, humans or machine active elements, which have power of attorney from the next higher-level process in the nest. The role of a process agent is to look after the interests of the client, just as real estate agents do for their clients. If the rules for nested processes are not adhered to, it is possible to arrive at a point where one can neither continue nor return to a prior point of acceptability. Such a state of affairs implies loss of control and therefore of integrity.

Figure 3 Spheres of control over process atomicity and commitment (multinode processing)



3A. Node structure (communication) 3B. Control structure (processing)
A node is a processing resource, man or machine. A sphere of control over process commitment encompasses all work, man or machine, on behalf of the initiating man or machine.

controlled dependency

Sometimes when the output of one process provides input to a subsequent process, the initial output is available prior to completion of the creating process. In such cases, it is logically required that the output of the creating process be contained (not depended upon) until after the point of commitment. That point is at least no earlier than the end of the highest-level atomic process in the nest, although it may be much later for management reasons.

The situation described above leads to poor performance for the simple reason that there is minimal concurrent processing. In fact, it requires strict sequencing of dependent processes. The best performance is achievable only when a process begins as soon as its resources are available without deadlock.

To begin a process earlier than is logically correct (from a commitment point of view) requires that the controlling element extend its influence to dependencies that are not yet committable. Consider the payroll application mentioned earlier, in which paychecks, transaction histories, and stock purchase deductions are the output of a process. Some subsequent process (usually the next) enters the stock purchase deductions and determines whether there is enough money in the account to purchase a share of stock. If so, a purchase transaction is created. While these processes may be logically sequential—that is, not in parallel—it is clear that considerable parallelism is possible and frequently desirable to meet a deadline.

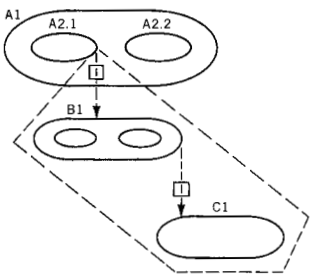


Figure 4 Spheres of control over noncommittable dependencies

Allowing parallelism while preserving back-out rights results in controlled dependencies. A dependent is allowed whenever the depender guarantees the ability to back-out and the creator of resources guarantees probability of resource stability equal to or greater than that required.

This form of logically sequential but physically parallel processing is typical of many applications in commercial data processing. An example is illustrated in Figure 4, where the dotted line represents the containment of dependencies on a result of process A2.1. The concept can also be thought of as a boundary of control over commitment that is grown dynamically to include dependent processes started earlier in process time, resulting in maximal parallelism. Atomic process A1 creates (by the end of A2.1) the data representing stock purchase deductions. The disjoint atomic process B1 accepts as input a named version of these deductions and creates stock purchase transactions as appropriate.

The latter process in Figure 4 can run with the following constraints: It cannot hold and use (lock) a resource (value) that is to be updated by the process started earlier, and it cannot update a value that the process started earlier depends upon. This property has been called bi-phase processing.[2] Simply stated, no process can be allowed to commit its results with any degree of certainty greater than that of its input. Usually the results of multiprocessing or multiprogramming should be the same as the results that would be achieved if each transaction were processed alone, except as the results may be affected by the sequence of processing.

In summary, controlled dependency is control over the use of results of a process that cannot yet be committed. From a wholly logical point of view, such a concept usually is not necessary. Without the concept, however, all processes would have to be strictly sequential if one depended upon the output of another, and the resulting processing time might well be longer than would be practical.

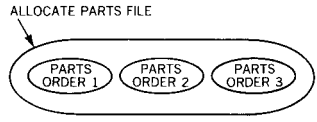
resource allocation

The sphere of control over resource (data) allocation is generally the same as that over process commitment. In some environments, however, a resource allocation sphere of control would produce excessive de-allocation and re-allocation, and processing performance would be degraded. To alleviate such degradation, it

is usual to establish a boundary of control not unlike that of commitment, but encompassing many commitments.

Figure 5 illustrates the point. Each of three units of processing labeled *parts order* is a boundary of control over an atomic process, or action, as well as over a commitment and over in-process recovery, all at the same time. Since all three require the same file, the sequential but otherwise unrelated processes are allocated the resource as though they were related. However, processing may terminate at the end of any boundary of commitment after each parts order is processed. The boundary of commitment, in this case, sometimes is referred to as a synchronization point.

Figure 5 Sphere of control over resource allocation



Boundary is around many disjoint actions, which, for purposes of performance, are considered a single unit of work (e.g., a batch job).

In summary, the resource allocation sphere of control is the assignment, or locking, of resources for a potentially greater period than is strictly required for correct processing. In particular, it is the grouping of unrelated processes for the purpose of saving the process time that otherwise would be required for resource unbinding and rebinding.

Recovery control

Aside from the requirement that the results of an operation be contained to preserve integrity, there exists the possibility that a user will decide that a previous course of action was inappropriate. The recovery operation is called in-process recovery if the action was taken prior to a point of commitment, and post-process recovery if the action already has been committed. Both in-process and post-process recovery spheres of control are application related.

If the results of processing are not yet committed, and allowance has been made only for machine errors that can be corrected by a rerun from a checkpoint, then the recovery operation is called system recovery and is not application related.

in-process recovery

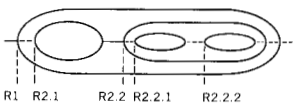
To preserve integrity it is necessary that an atomic process either not start at all, or start and finish in an acceptable manner. However, since atomic processes may be made up of other nested atomic processes, there are potentially as many places to return to for recovery as there are atomic processes in the nest.

This does not mean that if an atomic process cannot be completed satisfactorily, one must return to the beginning of the process. In fact, it is only minimally necessary to be able to back out to a point at or before the atomic process involved. Should there be a change of nodes involving long response time or high communication cost, however, it is usual to have a backup point for process-

ing every time a change of node occurs. Such a backup point is particularly important if more than one person is involved and one of them causes an error, making it necessary for another person to redo work already correctly completed.

Procedures are treated above as though they were always predefined, as in batch processing. But in the realm of interactive, on-line, or in-line processing, procedures may very well be dynamic because functions can be invoked in real time. With dynamic procedures, the boundaries constituting atomic processes and recoverable processes must be specified by the person who invokes a function. The boundaries are operators, or commands, which are in-line in the ongoing processing. Function keys on terminals are most often used for this purpose.

Figure 6 Spheres of control over in-process application recovery



Application may be backed out to points having semantic significance to the application. Subsequent processing may take a different path.
Each sphere of control must save information necessary to back out, since each is potentially transparent to the next higher level.

An example of the need for nested recoverable units of processing is illustrated in Figure 6. The work bounded by R1, initiated by a human operator, involves entry of a parts order which may be in error and have to be re-entered. R2.1 allows the erroneous portion of the parts order to be backed out. Subsequently, in the processing bounded by R2.2, the parts file is searched, and a set of choices evolves via the processing bounded by R2.2.1. An item in the parts order is found to be in short supply, and management approval is required before it can be shipped. The work done by the approving manager is bounded by R2.2.2. Should the manager discover in the middle of processing (decision making), that he has made an error, he requests backout of the process to the beginning of R2.2.2. (The callouts in Figure 6 indicate points where a command is given to initiate each sphere of control.)

Nested backout as described above is essential for the following reasons: Without it, the entire process would have to be backed out and the parts order re-entered. If the solution were to have many disjointed sequential processes, then a parts order discovered to be in error during processing could not be corrected because the user already would have entered it and received confirmation of the entry. In that case a formal cancellation transaction, complete with audit trail, followed by resubmission of a corrected parts order, would become the only realistic, and possibly legal, default.

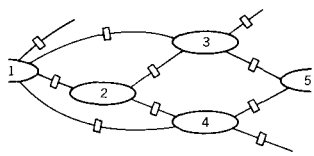
In summary, in-process recovery is control of the recording and subsequent use of whatever data is required to return to a previous point in the process—namely, the beginning of this recovery sphere of control.[3] Recovery spheres of control can be, and frequently are, nested. The spheres of control over process atomicity and in-process recovery often are boundaries for the same processing. To allow for procedure replacement, the boundary of recovery must coincide with the boundary of an atomic operator at some nest level, often in addition to the one at the highest level.

post-process recovery

The purpose of post-process recovery is to determine the source of, and correct for, an error discovered during processing but whose cause is no longer contained within an in-process recovery sphere of control. Four basic activities are necessary to recover from an error discovered after process completion. They are illustrated in Figure 7, where, first, a symptom of the error is detected in process 5. A check on the relation between two or more data items, for example, might reveal an invalid relation such as the quantity of parts received minus the quantity shipped not equaling the quantity on hand. Note that this is not the original error, but only a symptom or consequence of the error.

Second, the data elements believed to be involved in the error are brought into the sphere of control over processing established for post-process recovery. The process that created or last modified the data elements in question is determined from a journal. Input and output data from the past processes involved (3 and 4 in Figure 7) is brought into the post-process recovery sphere of control, which is nothing more than a dynamic atomic action sphere of control.

Figure 7 Sphere of control over post-process application recovery (action data dependencies)



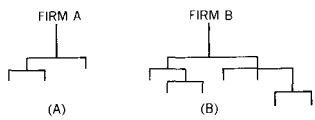
Numbers represent processes. Search backward to determine error source. Search forward to bound scope of dependencies.

Third, it is necessary to determine the extent of exposure resulting from the original error. This is accomplished by searching forward along the paths of dependency to identify those that depended directly upon the error—that is, those that depended upon a value that was wrong and now is right or thought to be right. There may be many such values and processes. For example, the wrong element may have been updated, resulting in two errors, each of which may have been depended upon by many processes.

Fourth, for each process that had different input, it must be decided whether the difference affected the outcome of that process. If so, then it must be decided whether to back out the old process and rerun to generate differences, or merely to compensate by means of another transaction.

The results of the fourth step are examined to see which output data would have been different, and the third and fourth steps are repeated until no more processes are affected. Once each process history is no longer needed, it is released from the post-processing recovery sphere of control so that other recovery procedures can use it, if necessary, provided that one is willing to commit to the recovery actions taken so far. Recovery processing is a normal process that must obey the same rules as a nonrecovery process.

Figure 8 Spheres of control over post-process application recovery (functional control dependencies)



The post-processing recovery sphere of control can be explained by using an analogy from management. Figures 8A and 8B illustrate the management hierarchies that might exist at two com-

panies. To recover from an error for which a given manager is responsible, that manager would first try to decommit the action. If that were not possible, he would try to compensate for the error. But if compensation were refused by the manager affected by the error, the problem would be escalated. Escalation always results in either decommitment or compensation, since at some level in the chain of command there is a manager whose responsibility includes both cause and effect.

To recover from an error for which no single manager is directly responsible, an attempt would be made to decommit or compensate for the error, as above, but if the managers involved could not reach agreement, then negotiations would be initiated to find a compromise. Should negotiation not be successful, the matter would be submitted to adjudication by the establishment of temporary common control. It should be noted that if only one party has kept records, they will be considered to be the facts. Therefore it is important for both parties to have complete records, which have the dual properties of being useful for recovery and necessary for audit.

In summary, post-processing recovery is control over processing that searches backward to find the source of an error, and then forward to bound the propagated effects of the error.[4] It is necessary that recovery be a normal process and that it contain all the relevant resources within its sphere of control over processing. Otherwise the network of resources would have to be locked as a unit, with the result that recovery could be the only active process in the entire network.

system recovery

System recovery is the restoring of a previously existing system state by establishing checkpoints that represent the earlier state. Checkpoints are often transparent to the process being checkpointed. Checkpoints are useful in either of two cases: The first case is when an application error occurs and no commitments or dependencies have been established since the last checkpoint, so that subsequent reprocessing can take a different path from that of the original processing. This procedure is nothing more than a transparent, in-process recovery sphere of control and is useful in a batch environment where the operator controls commitment.

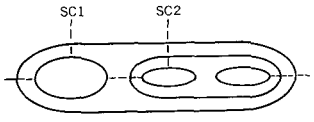
The second case is when, for example, a system fails because of a machine check, and the checkpoint represents a prior point from which processing will be repeated. This procedure is useful when recovery is not required to produce different results.

Unfortunately, the checkpoint philosophy does not allow for the replacement of an erring procedure, since the checkpoint can be taken anywhere relative to the boundaries of a procedure. If the checkpoint is taken at the boundary between two procedures,

atomic processes, or actions, it is really an in-process recovery sphere of control.

Figure 9 illustrates two system checkpoints, SC1 and SC2, at arbitrary points within an atomic process.

Figure 9 Spheres of control over system recovery



Controls unilateral back-out and (potential) rerun of uncommitted processing. Checkpoints (SC1, SC2) are applied to process, not machine.
Valid only for regenerating lost bits that have not been used to take a wrong path (wrong direction). Undoing of uncommitted processing does not require semantic knowledge.

In summary, system recovery is control over the recording and subsequent use, for recovery or system restarting, of at least the data that has been modified or newly created. Frequently, unmodified data also is recorded to save time on restarting. Since the system does not understand what the application did or meant to do, it can roll back processing only when no commitments have been made. This leads to the interesting dilemma of how the system knows when commitments have been made, since they are not recorded. And since no dependencies are maintained, any application that commits its results commits all applications, unless in-process recovery spheres of control are being used.

Audit control

The auditing of a process, or action, is the mechanism by which the validity of the process is determined. In any auditable system there must be a mechanism for remembering every action taken. Such a mechanism should be transparent to the process being audited. To audit a process, or action, means that one has defined the unit of process that constitutes the action. The kinds of auditing required are in-process immediately prior to process termination, and post-process as soon after process termination as possible. This bounding of auditable actions is described in terms of intraprocess and interprocess auditing spheres of control.

intraprocess auditing

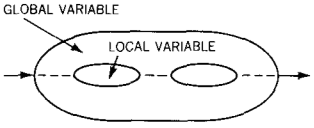
Intraprocess auditing validates the processing of single actions, such as translating a parts order into a shipping order. Single actions are those for which no point of commitment occurs except at the end of the highest-level atomic process.

There are two mechanisms for single-action auditing. One is the in-process audit, a procedure specified by the auditor and designed to catch undesirable process consequences. For example, a parts order for two items results in a shipping order for 20 items and the unit of issue is the same. However, if the parts order resulted in one item being shipped and the other back-ordered, any error is not one in which an external auditor would necessarily be interested. An internal auditor may be interested in it if it represents a deviation from policy. For example, it may be company policy to back-order all parts orders for quantities less than ten.

The other mechanism is the post-process audit, which requires complete reconstruction of the information necessary to deter-

mine what input was used, who initiated the action, when it was initiated, and what the result was. The post-process audit applies to both data and procedures, since both change over time. In this respect instruction data also has versions, and each version must be reconstructible for auditing or reprocessing. This is mentioned here for emphasis but is no different than the requirement levied against all data and its various versions.

Figure 10 Sphere of control over auditing (single action)



Who entered what data? When? What was the result of the action? Why (how)? That is, which version of what data and procedure was the basis for and consequence of what action?
Implicit is that auditable actions have identity.

Figure 10 illustrates two auditable subactions within an action. Note that it is necessary only to save or reconstruct global variables relative to the auditable action, since local variables would be recalculated.

However, if there are any separately and more detailed auditable subactions, the global variables relative to the subactions must be saved even though they are local to the larger action. Otherwise they must be reconstructible by an auditor-approved mechanism. Auditor approval is required for all reconstruction mechanisms; otherwise one could simply assert that a certain thing was true.

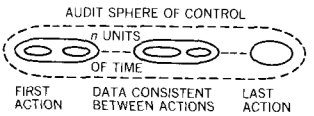
To the extent that an in-process recovery sphere of control has the same boundary as an intraprocess auditing sphere of control, the data that need be saved is the same. It will then be used by the post-process recovery function and the auditor.

In summary, intraprocess auditing is control over the recording of the input to, and output from, a process, together with the subsequent reconstruction of the information for the specific purpose of verifying and validating the original processing.[5] Intraprocess auditing validates the outcome of each transaction independently of the source or recipient of the transaction. Depending upon the nature of the system or application, the nested atomic processes may have to be independently auditable, in which case their input and output must be reconstructible.

interprocess auditing

Interprocess auditing validates the processing of an action that is disjointed as a function of time. It is the checking, tracing, and reconstruction of the various causes and effects of actions which are on behalf of an original action.

Figure 11 Spheres of control over auditing (across actions)



Trace of the disjoint actions performed on behalf of an original transaction.

Figure 11 illustrates three disjointed but related actions. They must be auditable as though no time had passed between them, even though the actions may have occurred months or even years apart. Interprocess auditing, like intraprocess auditing, has both in-process and post-process components. The in-process components are procedures, specified by the auditor, that are designed to catch undesirable process results relative to the original action, which may not be an error as a stand-alone action.

The post-process components are procedures, such as random samplings, that are used later to look back at a collection of actions and verify the original processing. This audit also requires the same knowledge as intraprocess auditing, along with the ability to tie related actions together, even though they may be disjointed as a function of time and of process, and even though nonrelated actions may occur between them.

To audit properly, within standards set up for the purpose,[6, 7] requires that all actions be audited or that a random selection of actions be audited, the selection being unknown ahead of time. Otherwise the audit could be invalidated by performing properly only those processes that it is known will be audited.

As a consequence, all information necessary for post-process auditing of any action must be saved at the time of the original processing. If the actions to be audited are not known at the time of processing, then all actions that might be audited must have their audit trail information established at the time of processing. An interactive procedure must be reconstructible. A procedure may be recorded each time it is executed, or it may be recorded only once and referred to by name upon each subsequent use. If the latter method is chosen, the process name must include the version, implicitly or explicitly.

In summary, interprocess auditing is the control over the recording and subsequent retrieval of the data necessary to verify the set of disjointed processes on behalf of an original transaction. Interprocess auditing validates the sequence and outcome of a set of related processes that are disjointed on the basis of time.

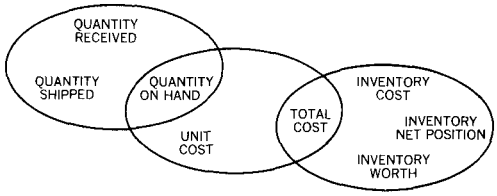
Relational integrity control

Relational integrity is the maintenance of related information in such a manner that a procedure receives the correct version of the related information it requires. This does not necessarily mean that the relations are correct at all times. It does mean that when a process needs them to be correct, the actions required for updating will have occurred. In addition, any process that attempts to create an invalid relation is prevented from doing so. The sphere of control over consistency is one of many mechanisms required to effect relational integrity.

consistency

The sphere of control over consistency includes the set of rules and assertions about a collection of related information. As pointed out by Eswaran et al.,[8] the assertions describe the relationships, consistency, and limits that must be maintained. The rules describe the conditions under which a procedure may acquire or create information or a subset.

Figure 12 Spheres of control over consistency



An update to an element of a sphere of consistency implies that no other update is possible by a disjoint action.
Spheres of consistency have versions.
Binding causes the enlargement of a sphere of consistency, or creation of another.

The example illustrated in Figure 12 shows three spheres of consistency. Sphere 1 asserts that the quantity received minus the quantity shipped equals the quantity on hand. Sphere 2 asserts that the on-hand quantity times the unit cost equals the total cost. Sphere 3 makes similar assertions, causing the field of information called *total cost* to lie in two spheres simultaneously.

The rules associated with a sphere of consistency describe, as a result of the assertions, the fields of information involved and the relationships that must be maintained for each procedure and by each procedure. Exclusive read or write allocations, via locking in all its forms, are various implementations of some of these relationship-preserving rules. The important point is that it is necessary for information to be consistent only when it is needed.

In summary, the sphere of control over consistency is control over the permissible uses of subsets of a collection of related resources. For example, the unit of issue, the quantity on hand, and the unit cost are related in such a way that if the unit of issue is to change, so must the unit cost. But if the unit cost is to change, the unit of issue does not have to change; in fact it should not.

Processing environments

The implementation of spheres of control depends on processing environments. For example, if a process is running in the batch mode, the spheres of control can share much the same information and control structures. However, with on-line or in-line applications, the various spheres of control usually must be implemented separately. In any event, they should be described separately to allow a choice.

Following is a description of the differences among the three primary processing environments: batch, on-line, and in-line.

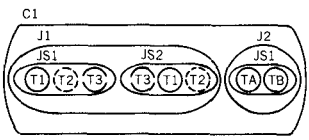
The more important characteristics of batch processing are:

batch processing

- A set of transactions is sorted to the same sequence as the files against which they are to be processed. The process (job) is broken up into smaller processes (job steps), each consisting of predefined procedures which are fixed for the duration of the job. Jobs by definition are unrelated to one another, even though they are executed in carefully defined sequences to prevent loss of integrity.
- No real-time variables can differ during a subsequent rerun. The entire process is repeatable without a journal of activity, provided that the files and file names have been saved. The files are saved by the simple and convenient expedient of not updating old files.
- The resources held under exclusive control (locked) are large—for example, files rather than records or fields. The response time is machine controlled since no human response time is part of the process, assuming that a fully automated storage subsystem is used.

Figure 13 illustrates a typical collection of spheres of control used in batch processing. For purposes of discussion we assume the use of an operating system such as OS/VS on System/370. The outermost sphere of control (C1) is for commitment control. Usually it is delineated by a (work) card and is enforced by exclusively allocating (locking) the resources used or created. Locking is performed by either the machine or an operator.

Figure 13 Batch processing—no human involvement



- Predefined procedure
- Predefined values for variables
- Repeatable without *journal*
- Resources allocated (locked) are large (files)
- Response time is machine controlled

Tx = *audit* JS = recovery
J = allocation C = commitment

The next nested spheres of control define the boundaries around two separate jobs (J1 and J2). Sequencing is specified and commitment controlled until the end of job 2. The spheres of control within each job are job steps, labeled JS1 and JS2 within J1, and JS1 within J2. They define the amount of processing performed as a result of an EXECUTE card. The spheres of control labeled T1, T2 · · · TB define subsets of processing on behalf of given transactions. They are units of processing for auditing purposes.

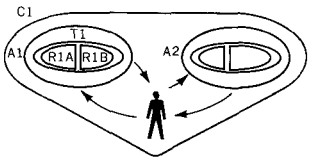
The more important characteristics of on-line processing are:

on-line processing

- Each transaction is processed alone. That is, there is not necessarily any attempt to batch more than one transaction in the same process. It is as though each transaction were a job step, and also a job if the data is unallocated between transactions. However, there is no human choice involved in the midst of processing as with in-line processing, discussed below. The procedures are fixed ahead of time; the only variables are the contents of each transaction, which are either acceptable and processed, or unacceptable, in which case the transaction is rejected.

- Since there is no attempt to establish a sequence for transactions, a journal is required for auditing and recovery in case reprocessing is needed. Each transaction consists of variables not known until the time of processing, so it is necessary to journal the entire transaction, not just the values of the variables it used or created.
- The resources locked are much smaller than in batch processing—records, for example, instead of files.
- Response time is under human control for as long as it takes to construct the transaction, and under machine control once the transaction has begun processing.
- Resources no longer needed may be released in the midst of processing, so long as it is still possible to recover from an error by backing out the entire transaction.

Figure 14 On-line processing—human involvement but not control



- Predefined procedure
- Dynamically defined values for variables
- Not repeatable without journal
- Resources allocated (locked) are medium size (records)
- Response time is machine controlled
- Unneeded resources sometimes de-allocated

Figure 14 illustrates a typical collection of spheres of control used in on-line processing. The outermost sphere (C1) defines the boundary of processing to be completed prior to making a commitment of the results. In this case, two transactions are to be considered as one transaction relative to commitment.

The spheres of control labeled A1 and A2 bound the unit of processing that is considered an action or process from the auditor's point of view. Within A1 is T1, a sphere of control bounding the unit of process from the user's point of view. Generally, this processing unit is called a transaction. Note that there is not human interaction within a transaction.

Within the processing of one transaction, the application may allow for rerunning part of a transaction to speed recovery from, for example, a machine check. Spheres of control R1A and R1B are examples of intratransaction recovery bounding.

in-line processing

The more important characteristics of in-line processing are:

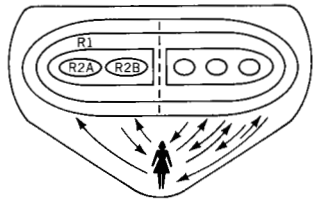
- Each transaction is constructed interactively, and its processing is directed by another active element, often a human operator, who may totally define the procedures in real time. Interactive problem solving and arbitrary queries and updates are examples.
- Almost all activity must be journaled for recovery and auditing because there is no other way of remembering a set of arbitrary variables and actions.
- An example of in-line processing is the query capability needed in an inventory control application when the query is not a predefined, and therefore programmed, set of code. It is typified by following a path through an associative network of relations, which may be represented as flat files with imbedded keys used for the associative linking. A journaling capability must exist to satisfy recovery and auditing require-

ments; otherwise one is limited to asking only those questions and causing only those results that have been anticipated and preprogrammed.

- The unit of resource that is locked is generally small, like a first, second, or third normal relation represented as a set of fields which are likely to be the concatenation of the subset of many records. Resources no longer required are generally de-allocated, again with the constraint that a backout of processing is still possible, or until it has been indicated that the highest level of commitment control has been terminated.
- In-line processing is typical for systems involving hypothesis trial and error techniques, which often result in backing up to one of *n* defined points and taking a different path. The backup points are dynamically defined by the terminal operator, often by the use of program function keys.

Figure 15 illustrates a typical collecton of spheres of control used with in-line processing. The outermost sphere, C1, again controls commitment. The first nested sphere, A1, is for auditing. The next nested sphere, T1, is the bounding of a transaction that a user interacts with and controls. That is, the user is an integral part of the process and may redirect it at any time. In contrast, in the batch and on-line environments the user is involved only at the beginning and end of a transaction.

Figure 15 In-line processing—human involvement and control



- Dynamically defined procedure (arbitrary)
- Dynamically defined values for variables
- Not repeatable without journal
- Resources allocated (locked) are small (fields)
- Process response time human controlled
- Unneeded resources usually de-allocated
- Hypothesis trial and error with frequent backup

The next nested sphere of control, R1, bounds that which the user may wish to back out and redo or just back out and forget, as when he discovers that the transaction was not necessary after all.

The next and last nested spheres of control, R2A and R2B, etc., are portions of the processing that the user may wish to back out and redo to a different conclusion. They are vital in any process subject to human error when the user wishes to redo only that portion of the process into which the error was introduced and those subsequently processed portions that depended on the backed-out, now erroneous, processing.

Summary

This paper describes most of the spheres of control required to maintain integrity in digital data processing applications. Among these spheres of control are:

- *Process atomicity*, which allows each function to be treated as a transformation without concern for the total inconsistency that otherwise would exist should a partial transformation occur. This is generally the unit of replaceable procedure. Process atomicity spheres of control are frequently nested.

- *Process commitment,* which bounds the dependencies and consequences of a function until such time as the work need not be undone or backed out and redone (to a potentially different conclusion). Process commitment spheres of control are frequently nested.
- *Controlled dependency,* which allows the controlled use of dependencies and updates to permit otherwise sequential processes to be partially parallel. Multiple active-element processes typically relate through controlled dependency when working toward a common goal.
- *Resource allocation,* which allows for better performance of some applications, to the potential detriment of others, by assigning some resources to a set of otherwise unrelated sequential processes.
- *In-process recovery,* which permits the return of a process to a previously acceptable point. This sphere of control also causes the recording of information required for post-process recovery, as described below. In-process recovery spheres of control are of necessity nested, unless, when an error occurs, one wants to redo all processing regardless of how recent or trivial the error.
- *Post-process recovery,* which consists in searching backward for sources of error, then searching forward to rerun or compensate for past processing or data-entry errors. This sphere of control is a process atomicity sphere of control that uses information recorded by and about previous spheres of control over process atomicity.
- *System recovery,* the establishing of and subsequent return to checkpoints. It has the disadvantage of resetting the system state including processes not in error. Further, it cannot be used if any commitments have been made since the last checkpoint, unless loss of integrity is not of concern. Hence its use generally is limited to single-thread batch processing. It is never used in interactive or multithread batch processing in which commitments have been made.
- *Intraprocess auditing,* which is the real-time validating of processing and the identification and recording of the particular processing for later random validation.
- *Interprocess auditing,* which is the validation, on behalf of an original action, of a sequence of related processes that may be disjointed in time. Usually they are selected at random after processing is complete.
- *Consistency,* which is the specification and maintenance of the relations expected by a set of application procedures.

Conclusions

The designers of the semantics of applications are concerned mainly with spheres of control. Concern for performance is a strong second. It is not unusual to eliminate automation of an

application if adequate performance can be achieved only by ignoring the spheres of control presented. The application is then done manually or not at all.

Computer scientists are striving for performance, sometimes at the expense of function. The lack of function frequently takes the form of not providing as much support for the various spheres of control as applications require. This apparent dichotomy leads to small, disjointed portions of applications being implemented with the required spheres of control accomplished manually outside the computer system.

Data processing is migrating toward on-line and in-line environments in which the data exists in the machine or not at all. Consequently, the persons who used to perform the functions required of the various spheres of control are no longer able to perform those functions. Therefore the spheres of control and their enforcement also must be automated.

Many of the procedures written for an application are intended to provide the auditing and control capabilities described in this paper. A standardized set of protocols permitted by these concepts could relieve application designers from having to program unique solutions for each and every program and application.

The most important conclusion is that automated digital data processing will not be successful in integrated or geographically distributed data-base applications unless the concepts and rules of the described spheres of control are adhered to. This conclusion is particularly relevant when one considers the increasing emphasis that auditors are placing on traceability and accountability in modern data processing systems, notably the real-time, on-line and in-line terminal-oriented systems being used more and more in the business community.

The ability to describe the boundaries of the various spheres of control should be added to the languages used to describe applications and direct the operation of computers.

ACKNOWLEDGMENTS

Thanks are due to L. A. Bjork since some of the recovery and auditing ideas presented here resulted from a recovery/integrity/auditing project in which we both participated, and which was managed by H. Herron.

I also owe thanks for their interest and encouragement to many friends, especially J. N. Gray, M. E. Senko, and I. L. Traiger of the IBM Research Division, J. Martin of the IBM Systems Research Institute, and Professor Brian Randell and his staff at the University of Newcastle-upon-Tyne, England.

CITED REFERENCES

1. C. T. Davies, Jr., *A Recovery/Integrity Architecture for a Data System*, Technical Report TR 02.528, IBM General Products Division, 5600 Cottle Road, San Jose, California 95193 (1972).
2. Personal communication with J. N. Gray, I. L. Traiger, and others of the IBM Research Division, San Jose, California.
3. B. Randell, P. A. Lee, and P. C. Treleaven, *Reliable Computing Systems*, Technical Report Series 102 (May 1977); also P. M. Merlin and B. Randell, *Consistent State Restoration in Distributed Systems*, Technical Report Series 113 (October 1977), Computing Laboratory, University of Newcastle-upon-Tyne, Newcastle-upon-Tyne NE1 7RU, England.
4. L. A. Bjork, Jr., and C. T. Davies, Jr., *The Semantics of the Preservation and Recovery of Integrity in a Data System*, Technical Report TR 02.540, IBM General Products Division, 5600 Cottle Road, San Jose, California 95193 (1972).
5. L. A. Bjork, Jr., "Generalized audit trail requirements and concepts for data base applications," *IBM Systems Journal* **14,** No. 3, 229–245 (1975).
6. "Advanced Electronic Data Processing Systems and the Auditor's Concerns," *Journal of Accountancy* **139,** No. 1, 66–72 (January 1975). This article is a preliminary report of the Auditing Advanced EDP Systems Task Force, Computer Services Division, American Institute of Certified Public Accountants.
7. *Systems Auditability and Control Study, Executive Report* (order number G320-5791), *Data Processing Control Practices Report* (order number G320-5792), and *Data Processing Audit Practices Report* (order number G320-5790), prepared for The Institute of Internal Auditors, Inc., Altamonte Springs, Florida, by Stanford Research Institute under a grant from IBM Corporation (1977).
8. K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, *On the Notions of Consistency and Predicate Locks in a Data Base System*, Research Report RJ 1487, IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (1974).

Reprint Order No. G321-5070.