Coordinated Resource Recovery in VM/ESA

by B. A. Maslak
J. M. Showalter
T. J. Szczygielski

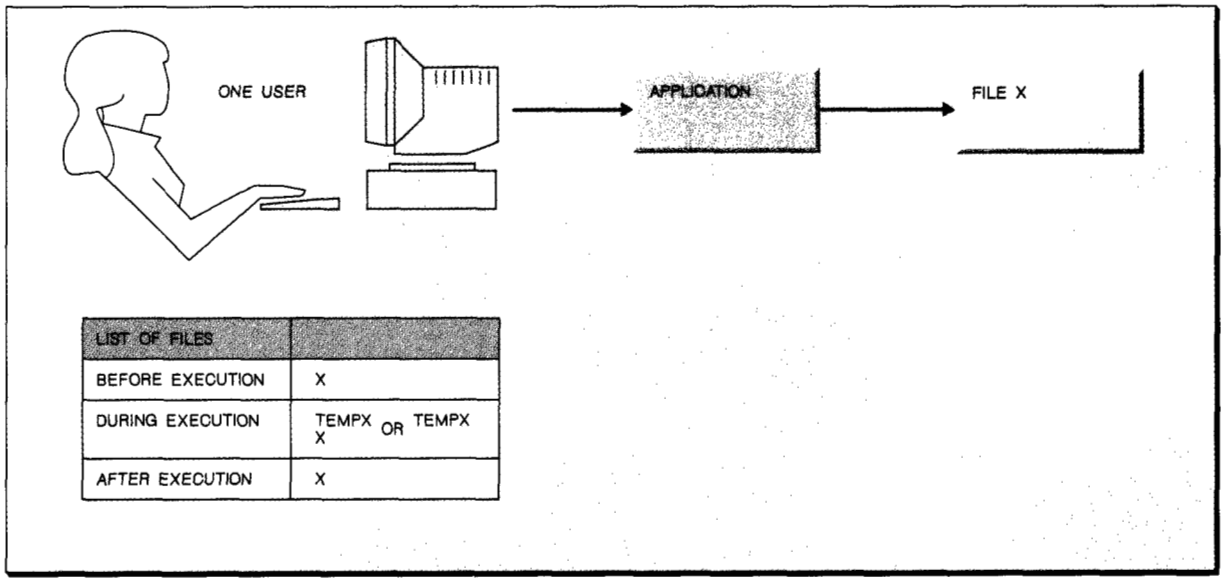
A system service for coordinated recovery of resources is a critical function needed for distributed processing environments because applications need to provide for data integrity while the location of the data and processes are transparent to the application. VM is the first IBM operating system to provide Coordinated Resource Recovery as a system service rather than as a service provided by unique environments running on the operating system, and the VM Common Programming Interface-Communications and Shared File System are the first subsystems to utilize the service. This paper is an overview of why and how VM provided Coordinated Resource Recovery (CRR). CRR is the implementation of the Systems Application Architecture™ (SAA™) resource recovery interface within Virtual Machine/Enterprise Systems Architecture™ (VM/ESA™). This coordinated sync point system service allows one or more applications or subsystems to update multiple resources and to request that all updates be committed or backed out together. The applications and their respective resources can be local or distributed. CRR either coordinates the request to commit or backout immediately, or supports automatic resource resynchronization in case a system or subsystem fails. When restart is not possible, CRR allows for intervention by a system operator or administrator.

Support has long been provided within IBM for the resource recovery requirements of database and data communication objects (e.g., relational tables, files, conversations) controlled by the subsystems IMS/VS, CICS, SQL/DS, and DB2™ in the transaction environments of the MVS/ESA™ and VSE operating systems.[1] The main benefit of resource recovery has been integrity for the supported data. However, two equally important and often overlooked results of resource recovery are application enablement and improved system management. Resource recovery simplifies application development because the application developer does not need to create recovery procedures, write documentation on how to use the procedures, or be aware of sharing and distribution characteristics. Resource recovery improves system management because uniform system-management recovery procedures can be developed by administrators of the resources.

For interactive or personal computing environments, if data are not shared and not distributed, application developers may provide for the data integrity needs of users. Since the data are private, no new system-management procedures are introduced for administrators. However, as data become shared and as data and processes are distributed, it is progressively more difficult for the application developer to support data integrity. When data integrity is supported by the application, unique system-management recovery procedures are introduced.

VM Coordinated Resource Recovery (CRR) extends the benefits of resource recovery (data integrity, application enablement, and system man-

Figure 1 Single user of a non–shared file



agement) to the Conversational Monitor System (CMS) environment of Virtual Machine/Enterprise Systems Architecture™ (VM/ESA™). The support is distributed, and interactive CMS is a distributed personal computing environment in which applications may use local or distributed shared resources and may cooperate with other applications. Thus, CRR is a model of the resource recovery solution for the evolving workstation-host environment called the enterprise local area network.

The first resource managers to use the support are in the VM Shared File System and VM Common Programming Interface-Communications. They provide immediate solutions for customers. Administrators are now able to move data from one Shared File System resource manager to another. CMS applications may cooperate with CICS/ESA™ applications so that each may update resources. For each of the above cases, data integrity is maintained, system management recovery procedures are uniform, and there is no impact to the application as sharing characteristics change or as data and processes are distributed.

Concepts of Coordinated Resource Recovery. In personal computing environments where sharing is not required, as shown in Figure 1, a single user invokes an application that updates a nonshared file, x. If the application developer has a data-integrity goal, the application contains a recovery procedure and provides documentation for the user about application failures. In this case, the recovery procedures and documentation are well understood and relatively straightforward. For example, the application may create the new level of a file as a temporary file, erase the old file, and rename the temporary file. Through this procedure, a consistent state of the file can be maintained.

Documentation should state what the user's tasks are if the application fails. For example, the user could be told to list the files. If only the temporary file is listed, the application makes the required updates and the user need only rename the temporary file. After an analysis by the user, the file could be restored to a consistent state—either the old state where the application needs to be reinvoked, or a new state where the application need not be reinvoked.

Figure 2 represents two users sharing files. Here documentation provided by the application developer is more complicated and recovery pro-

Figure 2 Multiple users sharing a file

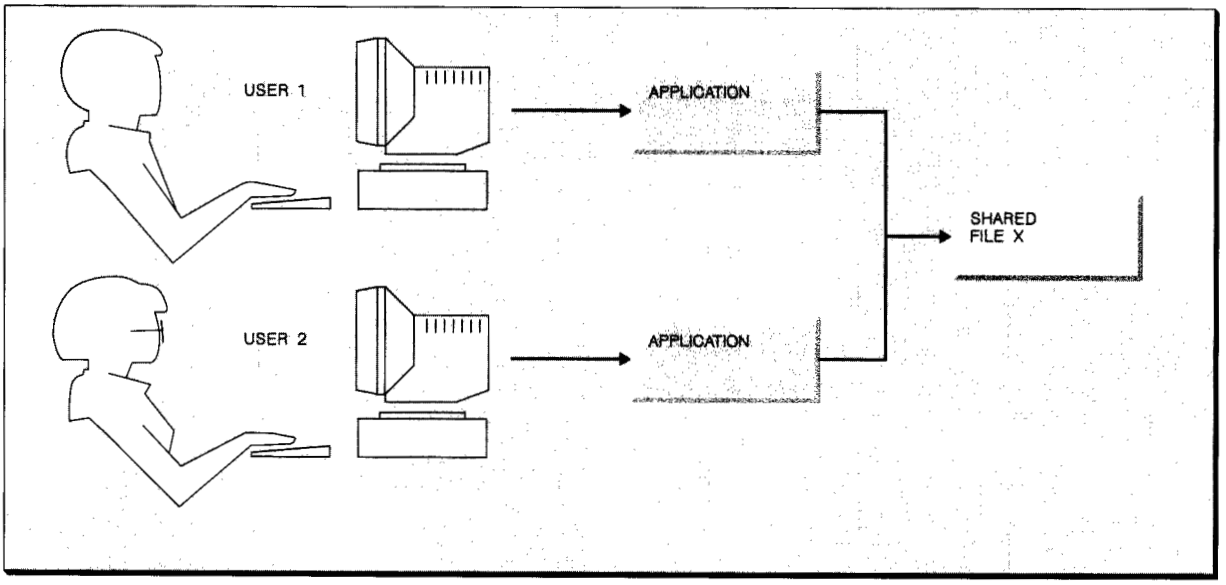
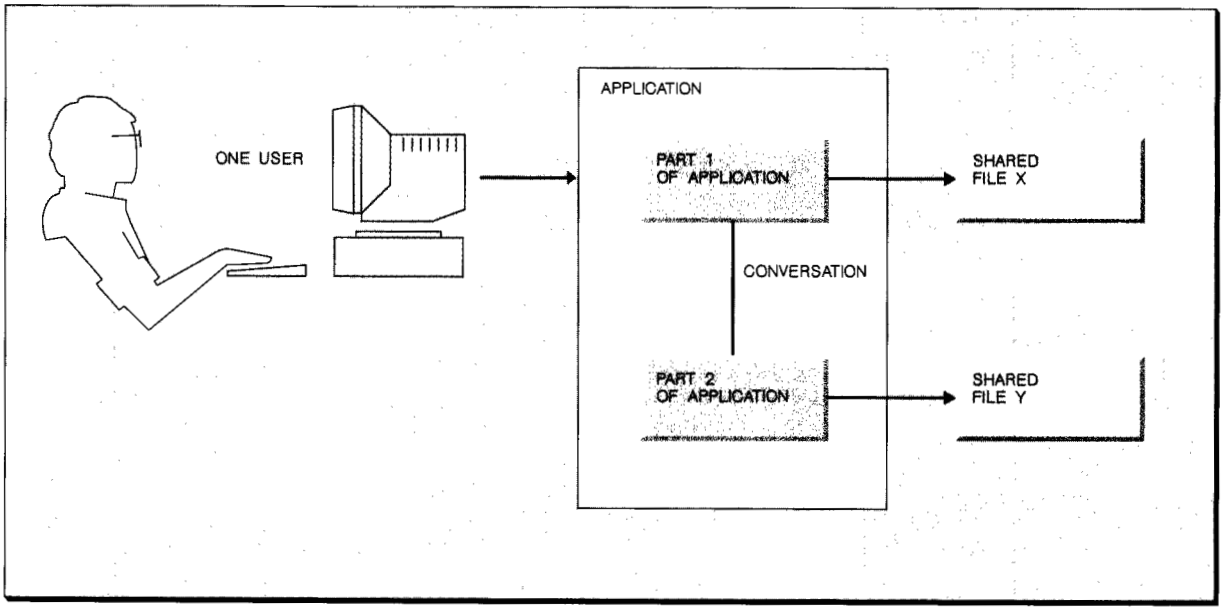


Figure 3 Distributed data and applications



cedures vary depending on the user. For the user who owns the file, the procedures could be the same as in Figure 1. All other users need to contact the owner, who becomes a data administrator and who can do an analysis similar to the analysis mentioned above.

Until the manual recovery procedure is done, new users may access inconsistent data unless very complex recovery and locking procedures are developed by the application developer. These procedures are further complicated because they may not execute in the user's personal computing environment—an environment that is subject to uncontrolled failures or shut downs. Again, a consistent state can be maintained but the burden on the application is increased. The developer requires a file system that supports data sharing with integrity, so that updates can be committed (*commit*) or be removed (*backout*) together, guaranteeing a consistent state.

Figures 1 and 2 deal with only one resource (i.e., a file). Now consider Figure 3, where the data and the application are distributed. A set of files makes up what the user may view as one file, but which is really two applications cooperating with a protected conversation. The application developer must handle the recovery of multiple resource types (i.e., the files and the conversations). It is even more difficult for the application developer to document what the end user or administrator needs to do in case of a failure. In spite of the system infrastructure for communications, the application developer may question if distribution is a good idea.

Finally, consider the example of an image application. A design of the application is shown in Figure 4 and has the following specifications:

- Forms are scanned by a device attached to a workstation and an image of the scan is saved as an object on optical devices attached to the workstation.
- A VM system with SQL/DS contains a master index that is the object catalog used to access the images of the forms.
- An MVS system with DB2 contains all the information extracted from the forms and is used for analysis of this information.

Application developers need a distributed system that supports a commit or backout that guarantees a consistent state for a set of heterogeneous distributed objects. Unfortunately, there is no such system. For the image application, the development team is required to design the recovery procedures. Extensive work will go into developing these procedures and testing them. More work will go into documentation of the procedures for users and data administrators. And, finally, the procedures will be unique for this application.

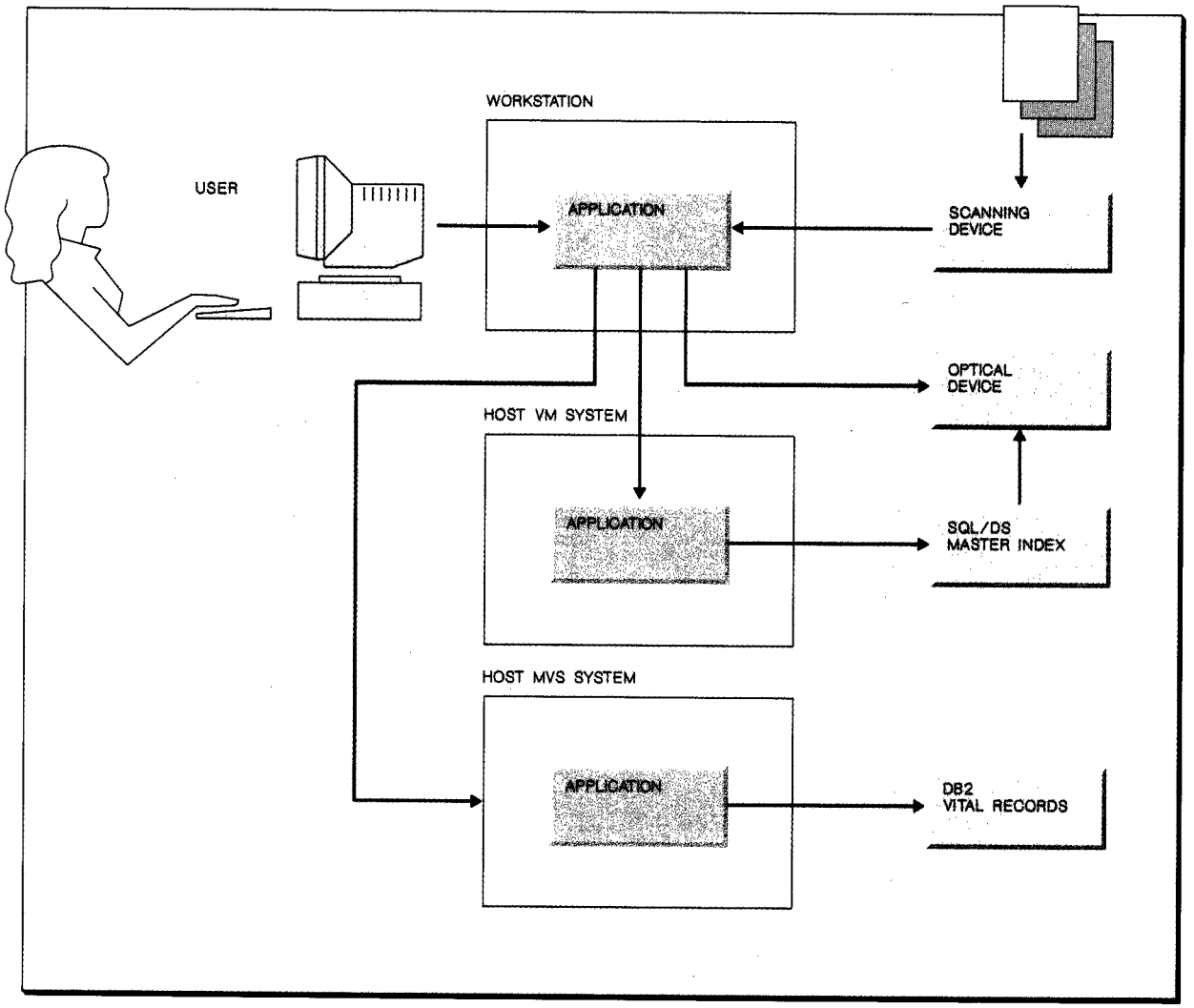
CRR provides commit and backout to CMS applications. These applications may be distributed, and the shared files are in the VM Shared File System. The support is extendable so other data management and data communication systems that execute on VM can take advantage of uniform system resource recovery procedures. The support shows that the synchronization point or recovery concepts of Systems Network Architecture are sufficient when a computing environment like CMS is connected to multiple local or remote resource managers. The support shows feasibility for the image application where the application and resources are distributed among a workstation and two heterogeneous host systems. Therefore, a distributed system can be built that will allow the image application developers to concentrate more completely on the evolving image technology rather than resource recovery issues.

Overview of CRR

CRR is based on concepts common to CICS/ESA, CICS/VSE™, IMS/VS, the Systems Network Architecture LU 6.2, and the SAA™ Resource Recovery Interface.

Briefly, CRR applications execute as a series of one or more logical units of work, where each unit consists of changes to *protected resources*. Protected resources are those resources that are subject to application-controlled synchronization (*sync point*) verbs, COMMIT and BACKOUT. Examples of protected resources could be database objects, conversations between cooperating applications, files, directories, messages, and queues. When the application issues the COMMIT verb, all of the protected resources are taken from one consistent state to another, so the logical unit of work appears to be atomic, relative to the protected resources. Failure in the process causes all of the protected resources to be returned to their

Figure 4 Image application example



last consistent state. The application may also issue the BACKOUT verb to request directly that the resources be returned to their last consistent state.

The COMMIT verb initiates a *two-phase commit* procedure, a prepare phase followed by a commit phase. During the prepare phase, each resource manager of a protected resource used by the application is polled by the system sync point manager to determine if all resources are ready for commit. A resource manager responds "ready" to the poll if the new state of the resource is on nonvolatile storage, and if the manager is able to promise that the commit can be completed if all other resource managers are ready. During the commit phase, all resource managers make the next consistent state of the resource available for applications or users. If the application uses the BACKOUT verb or if any one resource manager is not able to respond "ready," the system sync point manager tells all resource managers to make the last consistent state of the resource available.

While the system is processing the sync point verbs, any one or all of the distributed parts can

fail. For example, the CMS virtual machine, like a real workstation, can be terminated by a user; an operator may shut down a resource manager of protected resources; or a loss of power may cause system shutdown or communication failure. If a failure occurs, each logical unit of work that was affected by the failure will be resynchronized. This resynchronization happens as the parts (the resource managers that control the files or conversations) are restarted, and the result for each unit of work is that the protected resources are either taken from one consistent state to another or are returned to their last consistent state.

In general, CRR creates no new system operator or administrator tasks. However, until resynchronization is complete, the protected files remain locked. Thus, if some part is not restarted quickly (e.g., an operator misses a failure message or a communication medium needs to be repaired), users may request system operator or administrator intervention. In such conditions, CRR provides operator display commands so that, for each unit of work, it can be determined what restart actions are required.

In the extreme case that timely restarts cannot take place, CRR allows the system operator to make heuristic decisions, and allows the application developer to warn the operator about the consequences of such decisions. The warning information is provided by the application developer at execution time within an object called *transaction tag*. Heuristic decisions are made when users tell the operator that a resource is not available. These decisions are based on what the operator knows or is able to learn about the resources and application. Decisions are expressed in a command and can force a specific protected resource to its previous or next consistent state. A decision can break the atomicity relative to all protected resources involved in a unit of work. This breakage, known as *heuristic mixed*, is the condition where one or more resource managers commit the next consistent state and one or more backout to the last consistent state. The result is loss of integrity for a data object that consists of multiple distributed files. The transaction tags help the system operator to resist requests for heuristic decisions. For example, the tag could tell an operator to call a resource administrator. It could also inform the operator that an incorrect heuristic decision will require a rebuild of some distributed object, and that the rebuild will take much longer than the planned recovery procedure for broken media.

CRR system structure

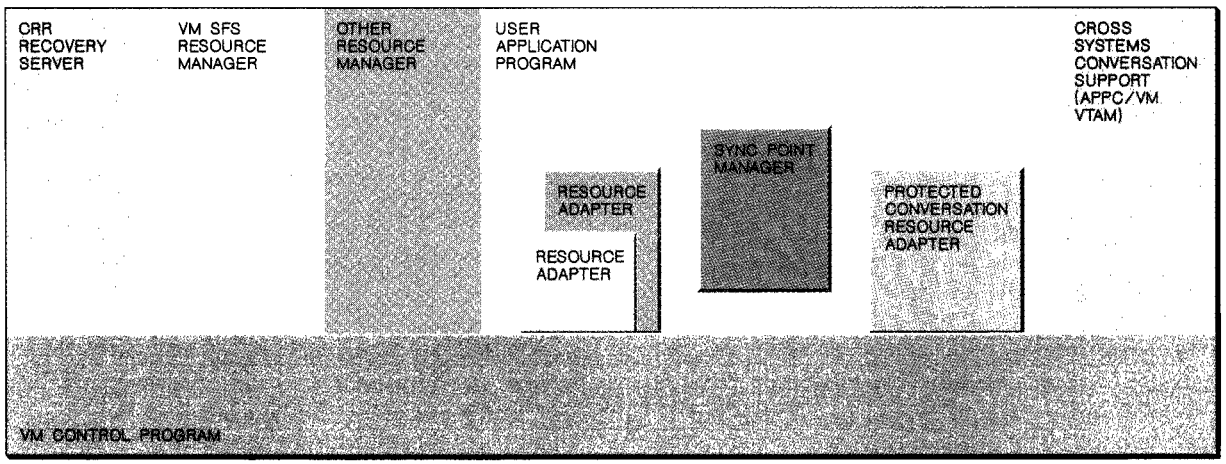
To support CRR and the SNA LU 6.2 sync point architecture in VM, several requirements had to be addressed. In VM, each CMS user has an environment and a virtual machine. CRR cannot require users to log on or dial into a special environment. Thus, system services that support CRR had to be available to any CMS application running in a user's virtual machine, and the recovery services of CRR are needed regardless of whether a user is logged on to the system. These services need to be invoked after a failure, even if the user that ran the original application does not log on again. From a system-management point of view, it is simpler if the recovery procedure is invoked automatically by the system. A single recovery log that is independent of resources provides a central point of failure control. As a result, CRR has to be provided partly in the user's virtual machine (commit coordination) and partly in a separate virtual machine (failure recovery). It was also desirable to provide a general capability to allow resource managers to participate in Coordinated Resource Recovery. Prior releases of VM did not have any notion of coordination of resource managers; each had its own independent recovery facilities with no capability to allow them to work together.

Coordinated Resource Recovery is provided to CMS virtual machines in VM/ESA through participating resources that provide a resource adapter and a resource manager, and CRR that provides a sync point manager and a recovery server.

Examples of these components are shown in Figure 5.

Resource adapter. Resource adapters execute in the user's virtual machine and provide the application programming interface to the resource manager. A resource adapter can be an elaborate linkage component that communicates with a distributed resource manager (e.g., advanced program-to-program communications, or APPC/VM) or a simple routine that calls a resource manager executing in the same environment as the application. A private protocol can be used between the resource adapter and the resource manager if the resource manager does not distribute to an-

Figure 5 VM/ESA structure for VM Coordinated Resource Recovery



other resource manager. (See the section Follow-on Considerations later in this paper.) If the resources are protected resources, the resource adapter interacts with the sync point manager. This interaction includes registration, handling various coordination exits that are invoked by the sync point manager, and returning status information to the sync point manager. In this case, the resource adapter is sometimes called a *protected resource adapter.*

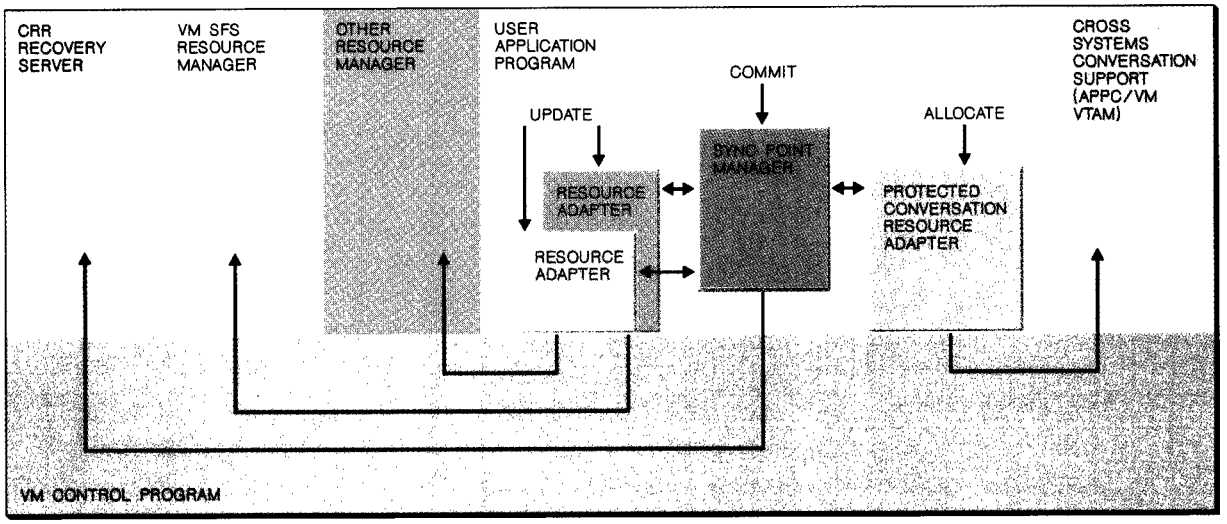
VM/ESA provides two protected resource adapters with the system, the Shared File System (SFS) protected resource adapter and an APPC/VM protected conversation resource adapter.

Protected resource adapters support CMS *work units*. A work unit is the application scope that controls which resources participate in a sync point. The application can specify which work unit is involved when changes to protected resources are made or when protected conversations are initiated.

CRR enhances the support for multiple work units in a single virtual machine. Specifically, applications, the sync point manager, and protected adapters can still support multiple concurrent work units. Each work unit can now support multiple protected resources in a single commit scope. Tying together the work units of two application environments with a protected conversation is also supported. Work units in different virtual machines that are related are identified by a common logical unit of work identifier (LUWID). This LUWID is the same as that defined by the SNA LU 6.2 sync point architecture and is assigned by (or received from) the initiating system. If the initiating system is a VM/ESA system, the LUWID is assigned by the CRR recovery server. Each work unit can have a series of sync points. A sync point addressed to a work unit (explicitly or by default) does not affect activity on other work units in the virtual machine. On the other hand, a sync point addressed to a work unit that is related (via the LUWID) to work in other virtual machines or other systems does affect that related work.

Resource manager. A resource manager is a system application running in a virtual machine that directly controls one or more VM resources. If the resources are shared among multiple users, the resource manager generally runs in a virtual machine that is separate from user virtual machines. This is sometimes called a server virtual machine. A resource adapter-resource manager pair is similar to the notion of a client-server pair. If the resource manager is controlling protected resources, it receives coordination information from its resource adapter. The resource manager establishes communication with its recovery server during resynchronization initialization and exchanges information that is required during resynchronization recovery.

Figure 6 VM/ESA calling structure for VM Coordinated Resource Recovery



VM/ESA provides a Shared File System protected resource manager. In addition, APPC/VM protected conversations are supported through the VM/ESA control program for local conversations and APPC/VM VTAM support for conversations that cross systems.

Sync point manager. The sync point manager is the part of the CRR facility for CMS that resides in the application's virtual machine. When a protected resource adapter is invoked by the application to update a resource, the protected resource adapter registers with the sync point manager to participate in CRR. The sync point manager coordinates the updating of protected resources and distributes the coordination of protected conversations to other sync point managers.

When the application issues a commit, the sync point manager controls synchronization point processing by invoking the participating resource adapters (i.e., those that have previously registered) through the following sync point manager exits:

- Precoordination—Checks participating resources to ensure they are ready for a sync point
- Coordination—Implements the one-phase and two-phase commit protocols. It is during this exit that the sync point manager invokes the recovery server to record the state of the sync point in its log.
- Postcoordination—Performs cleanup processing after a sync point

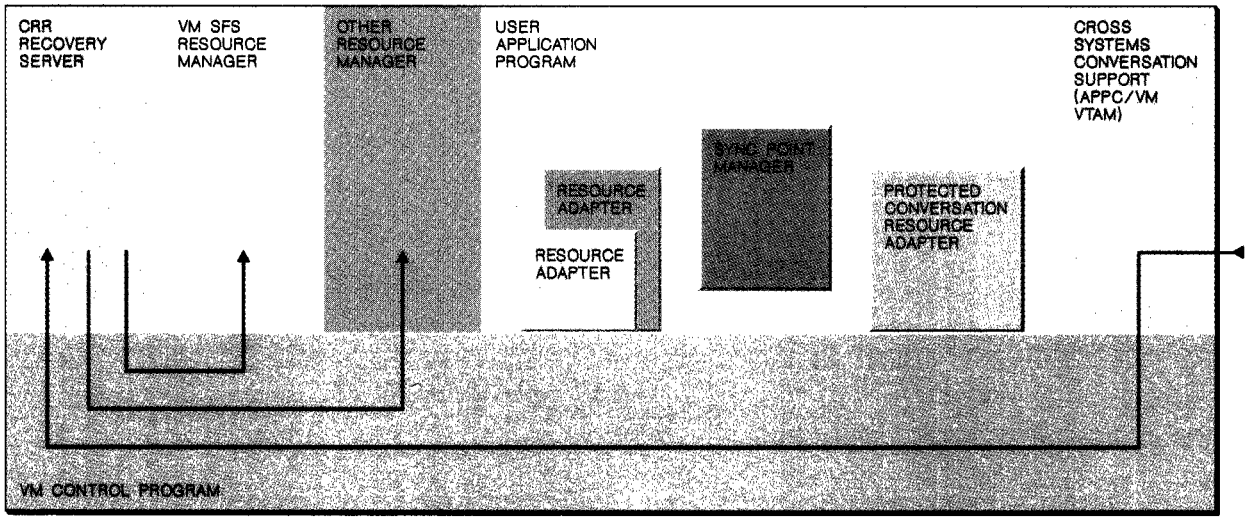
The following exits are not considered sync point exits:

- End of work unit—Does cleanup processing before the work unit ends
- Backout required—Puts the protected resource manager in a state such that roll back (backout) is required

Recovery server. The recovery server handles logging for the sync point manager. It receives information from participating resource managers during resynchronization initialization that is used in combination with the sync point manager log information if a resynchronization recovery situation occurs. The calling relationships are shown in Figure 6.

The recovery server also handles resynchronization for failing sync points. Resynchronization recovery is initiated on the system that originally issued the sync point. If this is a VM/ESA system, the recovery server initiates an Exchange-Log-Names/Compare-States transaction with the resource managers invoked from this system. This

Figure 7 Resynchronization recovery initiated from another system



includes APPC/VM VTAM support if a protected conversation has been allocated with another system. If another system originally issued the sync point, the recovery server receives an Exchange-Log-Names/Compare-States transaction through APPC/VM VTAM support (see Figure 7) and then initiates an Exchange-Log-Names/Compare-States transaction with the resource managers invoked from this system.

Local area network work groups

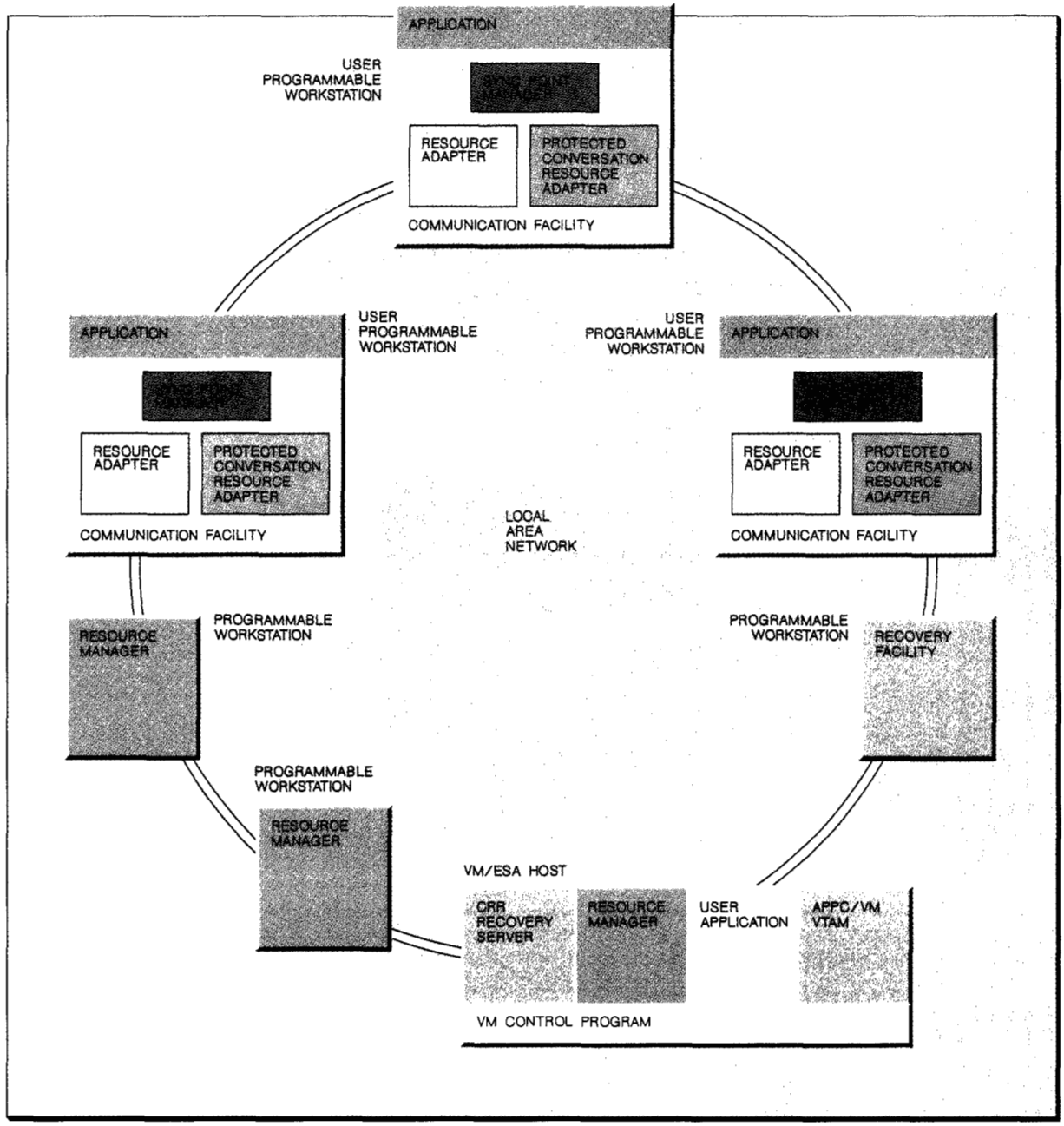
The virtual-machine structure of VM/ESA can apply as a model to the real machine structure of a programmable workstation (PWS) local area network (LAN) work group. Each user runs in a PWS rather than a user virtual machine. Each user PWS can contain a sync point manager that communicates with a separate recovery server PWS over the LAN. User PWSs can also contain protected resource adapters and a protected conversation resource adapter to support participation in Coordinated Resource Recovery. They also need communication facilities to support cross system conversations. The recovery server on a PWS can maintain a recovery server log like the VM/ESA recovery server. Each resource manager can run in a separate PWS with its own resource manager recovery log. This allows resynchronization recovery even if a user shuts the PWS off at the wrong time. A PWS structure of this type would also support the connection of a VM/ESA host with protected resources and a distributed application on the LAN. See Figure 8 for an example of this structure.

Applications and CRR

CRR, in combination with the resource managers that participate in its processing, lets the application developer concentrate on the best way to design the application without having to worry about data integrity. The application developer can decide that the best way to design the application is a cooperative model where pieces of the application run in different processors or virtual machines; the application can also interact with more than one data repository.

It was recognized during the design of CRR that applications vary in complexity. Some applications simply update a file, while some applications are complicated multiuser servers which manage the data. Other types of applications vary in their degree of complexity. The design of CRR and its interfaces takes into account the complexity of application designs, but does not make the simple application complex. CRR requires little to no interaction by an application, and provides interfaces when more interaction with CRR is needed by an application. These application interfaces

Figure 8 Local area network work group picture



can influence application performance, application error handling, and provide application-specific information for support personnel if a failure occurs during CRR processing.

Protected resource managers and CRR. The initial release of VM/ESA provides two resource managers that participate in CRR processing: the Shared File System and APPC/VM conversations. These

resource managers and any other resource managers that participate in CRR processing are called *protected resource managers*. CRR processing involves one or more applications updating protected resource managers' objects and requesting that all the updates be committed or rolled back to their last commit point. CRR must be told which resource managers accessed by an application are protected. When deciding how this was to be done, two choices were looked at. One was to have the application identify each resource manager to CRR; the other was to have each resource manager identify itself to CRR. Related to that was a question of how and when data (or other objects) owned by a resource manager became protected. Since data are a corporate asset and since data resource managers have their own philosophy on data sharing and data authorization, CRR requires that each protected resource manager (not the VM/ESA operating system, the application writer, or application user) decides when and how it becomes protected. Having the resource manager interact with CRR keeps the control of the data (object) with the resource manager and the complexity of interacting with CRR out of the application logic. Each resource manager defines what if anything an application must do to cause the resource manager and its objects to be protected.

Generally, an application first interacts with CRR by issuing a request to a resource manager. The resource manager interacts with CRR on the application's behalf. So, for example, an application issuing a request to the Shared File System causes the Shared File System to interact with CRR and become protected. The Shared File System also allows the application to have special types of files, called nonrecoverable files, that are not protected. Reference 2 explains this in more detail. As another example, if an application desires a protected conversation, it indicates so with a synchronization parameter on the APPC/VM or the Common Programming Interface-Communications application programming interface. Each resource manager that participates in CRR processing will decide whether or not the application program has to explicitly ask for the resource manager to be involved in CRR processing for the logical unit of work.

Requesting CRR commit/backout. An application makes changes to objects owned by resource managers. When the objects changed are protected, changes are not made permanent until requested by the application. Similarly, the application can request that the changes are backed out to the last commit point.

The first release of the VM/ESA operating system provides three different ways to make these requests. For new applications, CRR provides the application with SAA Resource Recovery Interface services. The application calls the commit service to make the changes to all protected resource managers' objects permanent and the backout service to roll back the changes to the last commit point. The result of the Resource Recovery Interface call is returned to the application.

The Shared File System (SFS) in the VM/SP operating system provides commit and backout interfaces for SFS files; but to provide migration and coexistence to the VM/ESA operating system for existing applications that use this interface, the SFS commit and backout services will call CRR processing when invoked. The outcome of the coordinated commit and more detailed error information for the SFS resource manager is returned to the application. This is the second way for applications to commit or backout changes to all protected resource managers' objects.

The third method is an implicit action by the VM/ESA operating system. If the application updates protected resource managers' objects and returns to the operating system without having invoked CRR commit or backout services, CMS issues a CRR commit to all protected resource managers on the application's behalf. If the application ends abnormally, CMS issues a CRR rollback to all protected resource managers on the application's behalf. Implicit action by the operating system provides ease of using CRR and is also a migration and compatability consideration for existing applications running on the VM/SP operating system using the SFS. Implicit action is also taken by CMS for SFS files in the VM/SP operating system.

Customizing CRR. Although explicit CRR application action is recommended, from the previous discussion one can see that in the simplest case, an application can update protected resource managers' objects and end without issuing any direct calls to CRR. CMS will then take implicit CRR action.

Because of the structure of CRR and because of the sync point architecture CRR follows, CRR has a set of default operational characteristics. Some applications, however, might want to change some or all of these default characteristics and CRR provides a service to change the following defaults:

- Waiting for resynchronization
- CRR sync point manager communication with its recovery server
- Heuristic decision when abnormal failure occurs

Waiting for resynchronization (recovery). If a failure develops during CRR commit processing, the SNA LU 6.2 sync point architecture stipulates that as a default action, CRR completes recovery with all failed protected resource managers before returning to the application. If the application runs in a batch environment, the application would probably be written to use the CRR default, or to wait for recovery to complete before CRR returns to the application. However, if the application runs in an interactive environment, it probably would not be practical to wait until recovery is complete. In this case, the application programmer would want to have CRR return to the application after an attempt at recovery is made. CRR allows an application to override the default and request CRR to return to the application after attempting recovery with all failed protected resource managers. CRR provides recovery with these failed protected resource managers later asynchronously. In the former case, the application is guaranteed to know the outcome of the commit request. In the latter case, the CMS interactive environment becomes available for the application to complete. The application learns the result of the commit request and that the commit processing has not yet completed. This lets the end user start more work in its interactive environment or to log off.

Sync point processing runs in the application's virtual machine and its recovery and log manager runs in a separate CRR recovery machine.

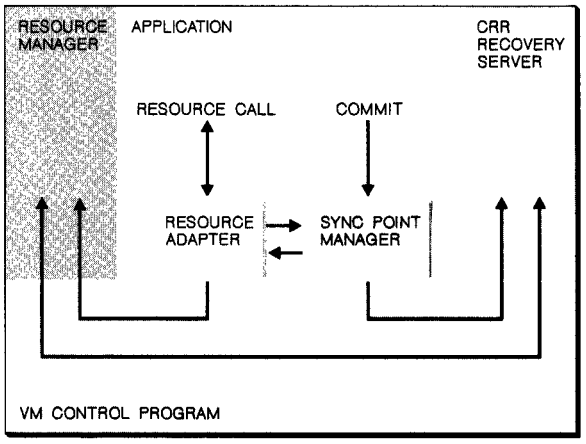
CRR sync point manager communication with its recovery server. As explained earlier, CRR's sync point processing runs in the application's virtual machine and its recovery and log manager runs in a separate CRR recovery machine. During CRR processing there is communication between the two pieces.

When an application issues a commit or starts protected work for the first time, the CRR sync point manager must communicate with its recovery server. Typically when this happens, the CRR sync point manager waits for a response from its recovery server before other processing can continue. This is the CRR default mode of operation. For most applications, it is sufficient if the sync point manager waits until the recovery server completes its request. However, there are some applications that may use CRR, such as a server machine, that could overlap activities if CRR allowed other processing to continue before the recovery server completes the CRR request. CRR allows this default option to be changed. The application can request that the CRR sync point manager allow other processing to continue before processing with the recovery server is complete and avoid serialization in the application. A server application, for example, can dispatch other server work. The sync point operation is redispatched by the server when CRR indicates its work has completed.

Heuristic decision when abnormal failure occurs. In rare cases, an abnormal failure occurs during CRR commit processing causing CRR to make a heuristic decision (to backout or to commit). In cases such as protocol violations between distributed sync point managers, CRR makes a decision because the decision maker caused the abnormal failure. CRR has a default heuristic decision which is to roll back all changes. The application can change this predetermined heuristic decision action to commit all changes.

Giving help if a failure occurs. CRR commit processing usually completes without errors. However, a failure can occur. A failure will cause recovery to take place. A failure can occur when the application developer or the application user is not available. Depending on where a failure oc-

Figure 9 Resource adapter and resource manager interaction with CRR



curred in the CRR processing, objects owned by protected resource managers may be unavailable until recovery takes place. Support personnel, however, may receive requests for manual intervention in the CRR process to make the objects available again. The application is not involved in recovery, but to help in the recovery process when manual intervention is requested, an application can give the CRR sync point manager information to store on the CRR log. This information, called a transaction tag, can tell the CRR support personnel the actions to take for this logical unit of work if a recovery condition occurs. For more information on transaction tags, see Reference 3.

Problems during CRR processing. If an application issues a commit request to CRR, the commit may not complete successfully or completely. Since CRR is a global commit of all protected resource managers involved in the logical unit of work, the return code to the CRR commit and backout function indicates the result of the coordination of all registered resource managers but does not give separate indications for each. Some applications desire additional information about each protected resource manager. The application may use this information to correct the problem that caused the failure; another use could be to provide information for support personnel in a common format and then end the application. CRR provides a service for an application programmer to access the CRR error information optionally provided by each protected resource manager. In the VM/ESA operating system, both the Shared File System and APPC/VM protected conversations provide CRR error information. CRR error information provided by the VM/ESA operating system protected resource managers are documented and a routine to format the content of the error information is also available. The format of the error information does not have a defined architecture.

Resource managers and CRR

As described earlier, applications invoke protected resource managers and the protected resource managers interact with CRR. CRR provides the rules and interfaces that resource managers must follow and use, allowing any resource manager to become protected. The resource manager implements these rules according to its design and processing.

As a background to understanding the CRR rules and interfaces for protected resource managers, it is important to understand the typical VM/ESA operating system resource manager structure for which they were designed. Figure 9 shows a user application, a resource manager, and a CRR recovery server. It further shows a functional split between the resource manager and a piece of its code that resides in the user's virtual machine, called the resource adapter. The application in the user machine makes a resource request. This request is handled by the resource adapter, which sends the request to the resource manager for processing. The resource adapter returns the result of the request to the application. Resource managers that have a different structure can determine how their resource managers functionally relate to the one described in Figure 9. The figure further shows the resource adapter's interaction with the CRR sync point manager residing in the application's virtual machine, where the CRR recovery server interacts with both the CRR sync point manager and the resource manager. Finally, note the application's commit request directly interacts with the CRR sync point manager.

Protected resource managers interact with CRR for three main functions: registration, coordination, and recovery. A discussion of each follows along with discussions of resynchronization, exits, multitasking servers, and support personnel help.

Registration. As mentioned earlier, when a protected resource manager is invoked by an application, it is the responsibility of the resource manager to interact with CRR. CRR provides a registration service. Registration is the method whereby CRR is informed that a resource manager is protected. Resource managers are required to register with CRR the first time the protected resource manager is invoked by an application and can unregister when the resource manager is no longer needed, typically when the application ends. This registration process is performed by the resource adapter on behalf of the resource manager in the virtual machine where the application is running.

CRR requires specific information from registering resource adapters for two purposes. The first is to know the resource name and the CRR exits for which the resource manager is to be called when processing these exits. The second reason for requiring specific information is for recovery if a failure occurs during CRR commit processing. Therefore, as part of registration or in subsequent registration update calls, the resource adapter supplies information needed by CRR for coordination and recovery. When CRR processes a commit request, it writes a log entry that lists all protected resource managers involved and information necessary to contact that resource manager should a failure occur during the commit process.

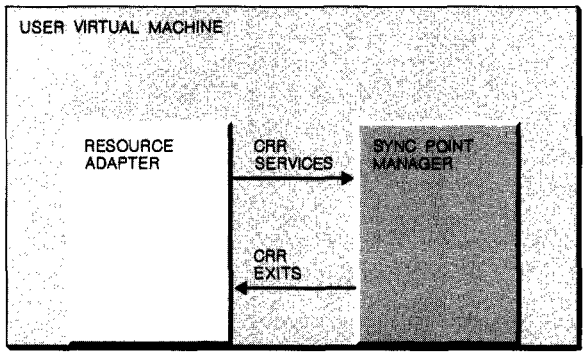
To provide for efficient CRR processing, the resource adapter indicates if it is updating its resource manager and if its resource manager can support anything besides a two-phase commit. Normally, a resource adapter indicates it has made updates (writes) to objects controlled by this resource manager. If write mode is indicated, CRR coordination processing performs logging for this resource adapter, and if a problem occurs during coordination, recovery will be done. The design of CRR is also sensitive to resource managers that want the performance benefit of a one-phase commit call if no other protected resource manager is registered. CRR allows a resource adapter that is not representing a protected conversation to indicate that it supports a simple commit call. If this is indicated and there are no other resource adapters registered for write mode, CRR coordination calls this registered adapter with a one-phase commit rather than a two-phase commit. No logging is done either by CRR or the resource manager. Because no logging is done in this case, a resource adapter cannot have a resource manager that distributes resource requests to another resource manager for the same logical unit of work.

CRR's design is also sensitive to resource adapters working for a back-level resource manager, such as a VM/SP operating system SFS server, which does not contain support for a two-phase commit. In such a case, the resource adapter indicates it only supports a one-phase commit. (A one-phase commit indication means no other resource adapter can be registered for write mode.) If indicated and the resource adapter also indicates that it is in write mode, CRR will deny the registration if there are other writers in this unit of work. CRR will deny subsequent registrations of other resource adapters if they indicate they are writers or try to change their write-mode setting.

An alternative approach considered in the CRR design required all protected resource managers, which might be accessed by an application, to register with CRR when the virtual machine first becomes active and to stay in the list until the virtual machine becomes inactive. This was viewed as more costly in terms of performance since the total registration list had to be scanned for every call, even if only one resource manager had been accessed by the application. The design of CRR did recognize, however, that an application may selectively request updates for protected resource managers' objects. For example, an application may update objects in three protected resource managers, issue a CRR commit; update two of the same three objects, and issue another CRR commit. CRR allows most registered resource adapters to suspend its active participation in CRR processing until the application updates the protected resource manager's object again. At that time the resource adapter can resume the active registration. This support provides an efficient method of suspending and resuming participation in CRR without going through the higher overhead of registration and unregistration. Effective use of this function can yield better performance.

Resynchronization initialization. As shown in Figure 9, the resource manager resides in a separate virtual machine. Resynchronization initialization takes place between the resource manager virtual machine and the CRR recovery server virtual

Figure 10 Resource adapter and CRR sync point manager interaction



machine. Resynchronization initialization is a function initiated by a resource manager after it becomes available within a processor but before the first CRR commit is invoked. For later recovery to be possible, the CRR recovery server and the resource manager must use consistent levels of log information between them. To ensure consistent log levels between a resource manager and the CRR recovery server, the resource manager must exchange log names with the CRR recovery server where the application invoking the resource manager is running. This exchange must be done before a CRR commit service is requested. To do this, the resource manager needs information from its resource adapter.

The resource adapter uses a CRR service to obtain the name of the CRR log and the name of the CRR recovery server. This information is sent by the resource adapter to its resource manager where it is used to determine when to initialize an exchange of log names with CRR (recovery server) and how to connect to the CRR recovery server for the log name exchange. This interaction is in the form of an APPC/VM transaction.

Coordination. Once a protected resource manager is registered with CRR, CRR calls the resource managers registered for a service when the service is invoked by an application or the VM/ESA operating system. Figure 10 shows this interaction. CRR calls resource managers registered for specific processing. When invoked by CRR, the exits provide an opportunity for functional customization by registered resource adapters.

The main exits taken are those invoked when an application requests the CRR commit service. Recognizing that resource adapters may have some processing to complete before and after its two-phase commit processing, CRR provides precoordination and postcoordination exits as well as coordination exits. A precoordination exit call gives the resource adapter a chance to do processing (for example, flush buffers) prior to CRR invoking registered resource managers for the actual commit or backout processing. A postcoordination call gives the resource adapter a chance to do necessary processing (for example, free buffers) before the application receives control.

The main exits for CRR are for coordination. Both the resource adapter and resource manager must follow rules established by CRR. The first coordination exit is for the *prepare-to-commit* function. At this time the resource manager is in a state from which it can either commit or backout its updates. A resource adapter that is not representing a protected conversation passes the current LUWID provided by CRR to its respective resource manager. The resource manager must write the LUWID and its recovery token (i.e., its specific recovery information) in its log when invoked by the resource adapter for a prepare-to-commit call. After a positive reply to a prepare-to-commit call, the resource manager gets either a commit or a backout call. If it receives a commit call, the resource adapter signals its resource manager to commit the changes to the object. If it receives a backout call, the resource adapter informs its resource manager to backout the changes to the object.

Recovery. If a failure occurs during CRR's commit processing, CRR automatically invokes recovery processing. The application program and the resource adapter are not involved. Recovery takes place between the CRR recovery server and the resource manager. This allows recovery to take place if the virtual machine where the failure occurred is not active and also allows the virtual machine to continue active processing if recovery cannot take place. For protected resource managers, CRR defines a recovery interface and rules for using that interface. If CRR's recovery server processes a failure that occurs during CRR's coordination processing, the CRR recovery server starts an APPC/VM transaction with the resource manager's recovery program using information passed on registration by the resource adapter

and subsequently written by CRR to its log. The CRR recovery server passes its log name, the LUWID (recovery token), and the action CRR wants the resource manager to perform for that LUWID—that is, commit or backout. CRR expects the resource manager to pass back its log name as well as the action the resource manager actually performed for the LUWID. Resource managers can allow operator intervention that manually forces work in doubt to be committed or backed out before recovery takes place. The resource manager must remember what specific operator action occurred. This action, called a heuristic decision, must be reported during CRR's recovery processing.

Other CRR exits. While the coordination exits are the main calls from CRR, CRR also calls registered resource adapters if requested when a backout is required and when cleanup conditions arise. When a backout-required condition occurs, commit is not allowed. CRR enforces this. However, CRR recognizes that a resource adapter might also want to enforce this condition preventing an application from doing unnecessary work. Therefore, when CRR learns of the backout-required condition, CRR will inform any resource adapter registered for this exit. The resource adapter is free to enforce this condition in a resource-specific way.

CRR also recognizes that resource adapters might have special cleanup considerations, such as deallocating APPC/VM conversations. When informed of a cleanup event, CRR will call any resource adapter registered for cleanup activities. The resource adapter can then perform its cleanup processing, such as freeing buffers, deallocating APPC/VM conversations or unregistering from CRR.

Multitasking servers and CRR. CRR recognizes that its coordination services might be used by multitasking dispatchers, but that they need the capability to dispatch other work while CRR does its processing. For this reason, CRR allows asynchronous responses by resource adapters to its exit calls. It also provides multitasking dispatchers (for example, servers) with a replaceable wait routine that allows them to provide an asynchronous exit from CRR. The multitasking dispatcher can process other work while CRR waits for resource adapter responses to complete their work.

Help for system-support personnel. If the application, the system, or the communications fail during CRR processing, CRR automatically initiates recovery or handles a request from a distributed recovery processor for recovery processing. However, CRR's design is sensitive to situations when manual intervention is needed in this recovery process. In this case, the system operator and possibly the system administrator can intervene using CRR recovery server commands. For a more detailed description of this support, see Reference 3.

Follow-on considerations

During the initial design of CRR there was extensive interaction with many architecture, research, and development people who are familiar with resource recovery. This section outlines some of the key follow-on considerations that were discussed. Many of these considerations relate to extending the usage of resource recovery. The others relate to improvements to resource recovery itself.

Extending the usage. Resource recovery as a system service needs to be made available in every distributed application environment. Thus, for example, the application developer should be able to distribute the application between CMS and Operating System/2® and assume the same resource recovery benefits.

As an open system service, resource recovery should be supported for new resource types and communication application program interfaces. Potential protected resource manager types are remote procedure calls, messages, queues, and distribution service objects such as print and spool.

The resource recovery system service and communication resource managers should operate with networks operating under non-IBM standards that support resource recovery, such as Open Systems Interconnection (OSI).

For distributed data management, where one resource manager accesses one or more resource managers for the application, protected conversations should be used between the application client and protected resource managers. This exploitation of resource recovery may save the re-

source managers from the need to implement a unique resource recovery facility.

Media recovery for compound objects should support resource recovery. Users should be able to view an object that is made up of multiple file segments and database segments. They should be able to archive or backup an object and recall it without understanding the underlying structure. The system should handle the media recovery for the object and be able to take into consideration the base recovery mechanisms of heterogeneous resource managers.

A consistent process model for multitasking is necessary before there can be an architecture for resource recovery and multitasking. Without the model, each system and resource manager will develop its own resource recovery rules relative to multitasking.

Improvements to resource recovery. VM implementation of transaction tags should be added to the communication application program interfaces. See Reference 3 for more information on transaction tags.

CRR uses the general "presumed nothing" two-phased commit protocol. Improved and application-specific commit protocols, like "presumed abort," should be implemented. These are optimizations to the two-phase commit protocol that may be made applicable to certain classes of applications, for example, to an application that only reads data. (See Reference 4.)

Applications should have access to extended error passback. This is primarily for system management applications that could provide extended recovery. For example, an interactive application could conceivably react to an error by invoking a system-management application that does media recovery for a compound object.

Additional system management follow-on enhancements are mentioned in Reference 3.

Summary

This paper presents an overview of CRR. The key conclusions can be summarized as follows: Data integrity, application enablement, and uniform system management together can be extended from the traditional transaction environments to a personal computing environment like CMS. Making resource recovery a system service allows new protected data management or data communication resource managers to be added to VM or to other systems. The Systems Network Architecture for recovery is sufficient to allow the applications and protected resources to be distributed among a workstation and multiple heterogeneous host systems.

Cited references and note

1. Full names of the subsystems referred to in the text are Information Management System/Virtual Storage (IMS/VS), Customer Information Control System (CICS), Structured Query Language/Data System (SQL/DS), and DATABASE 2 (DB2). MVS/ESA stands for Multiple Virtual Storage/Enterprise Systems Architecture, and VSE, Virtual Storage Extended.
2. C. C. Barnes, A. Coleman, J. M. Showalter, and M. L. Walker, "VM/ESA Support for Coordinated Recovery of Files," *IBM Systems Journal* **30**, No. 1, 107–125 (1991, this issue).
3. R. B. Bennett, W. J. Bitner, M. A. Musa, and M. K. Ainsworth, "Systems Management for Coordinated Resource Recovery," *IBM Systems Journal* **30**, No. 1, 90–106 (1991, this issue).
4. C. Mohan, B. Lindsay, and R. Obermarck, "Transaction Management in the R* Distributed Data Base Management System," *ACM Transactions on Database Systems* **11**, No. 4 (December 1986).

General references

J. Gray, *Notes on Data Base Operating Systems*, Research Report RJ-2188, IBM Research Division, 650 Harry Road, San Jose, CA 95120 (February 3, 1978).

B. Lindsay, L. Haas, C. Mohan, P. Wilms, and R. Yost, "Computation and Communication in R*: A Distributed Database Manager," *Proceedings 9th ACM Symposium on Operating Systems Principles*, Bretton Woods (October 1983).

C. Mohan, and B. Lindsay, "Efficient Commit Protocols for the Tree of Processes Model of Distributed Transactions," *Proceedings 2nd ACM SIGACT/SIGOPS Symposium on Principles of Distributed Computing*, Montreal, Canada (August 1983).

SAA Common Programming Interface Communication Reference, SC26-4399, IBM Corporation; available through IBM branch offices.

Systems Network Architecture LU 6.2 Reference: Peer Protocols, SC31-6808, IBM Corporation; available through IBM branch offices.

VM/ESA CMS Administration Reference, SC24-5445, IBM Corporation; available through IBM branch offices.

VM/ESA CMS Application Development Guide, SC24-5450, IBM Corporation; available through IBM branch offices.

VM/ESA CMS Application Development Reference, SC24-5451, IBM Corporation; available through IBM branch offices.

VM/ESA CMS Planning and Administration Guide, SC24-5445, IBM Corporation; available through IBM branch offices.

VM/ESA General Information, GC24-5440, IBM Corporation; available through IBM branch offices.

Barbara A. Maslak *IBM Data Systems Division, Route 17C and Glendale Drive, Endicott, New York 13760.* Ms. Maslak received a B.S. in mathematics from Marywood College in 1972 and joined IBM in that year. She was involved in the development of the OS/Virtual System 1 operating system. Later she worked on various development projects including ISQL for SQL/DS on the DOS/VSE and VM operating systems, and QMF/VSE on the DOS/VSE operating system. Most recently she was involved with the system design of Coordinated Resource Recovery on the VM operating system. She is currently an advisory programmer in the VM/ESA System Design organization. Ms. Maslak received a First-Level Invention Award in 1990.

James M. Showalter *IBM Data Systems Division, Route 17C and Glendale Drive, Endicott, New York 13760.* Mr. Showalter received a B.S. in mathematics from Clarkson University in 1967 and joined IBM that year. Subsequently, he received an M.S. in computer and information science from Syracuse University in 1979. He has been involved in DOS/VSE and OS/Virtual System 1 development, VSAM and QMF for VSE, and SQL/DS development in VSE and the VM operating systems. Most recently, he was involved in the Shared File System and Coordinated Resource Recovery in the VM operating system. He is currently a senior programmer in VM/ESA system design. Mr. Showalter received Division Awards in 1980 and 1983 and a First-Level Invention Award in 1990.

Thomas J. Szczygielski *IBM Data Systems Division, Route 17C and Glendale Drive, Endicott, New York 13760.* Mr. Szczygielski received an M.A. in mathematics from Wayne State University in 1968 and joined IBM in that year. He was involved in the development of the job entry subsystem and job management for OS/Virtual System 1. Later he worked on the system designs for SQL/DS on the DOS/VSE and VM operating systems, and for the Shared File System and Coordinated Resource Recovery on the VM operating systems. He is currently a Senior Technical Staff Member working on the system design for VM/ESA. Mr. Szczygielski received an Outstanding Contribution Award for quality in the development of parts of the job entry subsystem in 1973, an Outstanding Innovation Award in 1981, and Corporate Award in 1987 for SQL/DS architecture and development, and a First-Level Invention Award in 1990.

Reprint Order No. G321-5424.