OPIC
Office de la propriété intellectuelle du Canada

CIPO
Canadian Intellectual Property Office

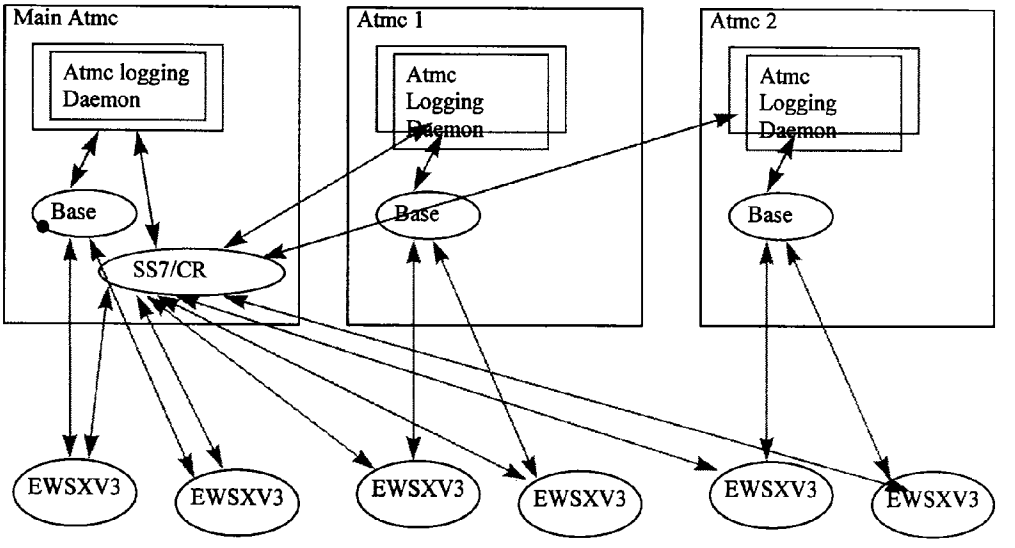
(12) (19) (CA) **Demande-Application**

(21) (A1) **2,221,661**
(22) 1997/11/20
(43) 1999/05/20

(72) ZELSTER, Rafael, CA
(72) BERTIN, Greg, CA
(72) HAMMERSCHMIDT, George, CA
(71) CROSSKEYS SYSTEMS CORPORATION, CA

(51) Int.Cl.[6] H04L 12/26

(54) **RECOUVREMENT DE TRANSACTIONS**
(54) **TRANSACTION ROLL FORWARD**



The Atmc Logging Daemon
1. Log transaction.
2. Play Back Transaction
3. Backup Switch
4. Restore Switch

Data Required For Logging and Playback
- Switch Address
- Time
- Application
- User Id
- Description
- Data

(57) In a method of monitoring transactions in a network application, a transaction log is maintained at a node, and a separate log is maintained. A playback is implemented which relies on the application to replay the original knowledge context of the transaction, and if this replayed mechanism fails, a transaction rollback mechanism is invoked to restore both the application and the node to a mutually consistent state.

ABSTRACT OF THE DISCLOSURE

In a method of monitoring transactions in a network application, a transaction log is maintained at a node, and a separate log is maintained. A playback is implemented which relies on the application to replay the original knowledge context of the transaction, and if this replayed mechanism fails, a transaction rollback mechanism is invoked to restore both the application and the node to a mutually consistent state.

TRANSACTION ROLL FORWARD

This invention relates to a method of monitoring network transactions, for example, in a wide area network controlled by network management software.

The Telecommunications Management Network (TMN) model, developed by the International Telecommunications Union (ITU), defines three areas: Element Management, Network Management and Service Management.

The Element Manager Layer controls the information exchange between network elements or groups of network elements. Functions within this layer communicate with the Network Management Layer by relaying element status and performance to a network management system (NMS).

The Network Management Layer in the TMN Model covers the end-to-end management of an entire network. It contains an overview of the network and knows how elements relate to each other. Focusing on network usage, traffic patterns, bandwidth availability and performance monitoring, the Network Management Layer interacts with the Service Management Layer by exchanging network status information for customer and service data. Responsibilities of the Network Management Layer include: controlling and coordinating the network view of all network elements within its domain, provisioning and modifying network capabilities for customer-service support, maintaining statistical, log and other data about the network and interacting with the Service Management, Layer on performance, usage and availability.

The Service Management Layer is the service provider's first point of contact with customers for all service transactions, including service requests and fault reporting, interaction with other service providers, and maintaining statistical data and interaction between services.

The present "state of the art" transaction roll-forward on a network element is to log all node (network element) affecting transactions from the last node backup. In the event of a failure all logs are re-applied to the node. A major shortcoming to this approach is the fact that the node has a limited context. The node logs only incorporates node affecting transaction not necessary element or network management transactions. For example an element management transaction "X" may be composed of node

transactions a, b and c. In the event of a failure the node will try and replay a, b and c. If a fails, b and c should not be applied because they are part of X which needs to be fully applied or not at all. The node logs have no knowledge of X. The result will be that the node and the applications supporting the node are in a inconsistent state.

An object of the invention is to overcome this problem.

According to the present invention there is provided a method of monitoring transactions in a network application, wherein a transaction log is maintained at a node, a separate log is maintained, a playback is implemented which relies on the application to replay the original knowledge context of the transaction, and if this replayed mechanism fails, a transaction rollback mechanism is invoked to restore both the application and the node to a mutually consistent state.

By implementing transaction logging our solution now maintains a log similar to the nodes. We then implement a playback which relies on the application to replay the original knowledge of the context of the transaction. If this "replayed" transaction fails than the applications transaction rollback mechanism is invoked and restores both the application and the node to a mutually consistent state.

By implementing centralized transaction logging our solution now maintains a log similar to the nodes. We then implement a playback which relies on the application to replay the original knowledge of the context of the transaction. If this "replayed" transaction fails than the applications transaction rollback mechanism is invoked and restores both the application and the node to a mutually consistent state. When we implement a playback we also ensure that the log is played back in a synchronized fashion to ensure node and application consistency.

The invention can be applied to any situation where a node is modified by a multi stage transaction and may at a later time need to be restored.

The invention will now be described in more detail, by way of example only, with reference to the accompanying drawings, in which:-

Figure 1 is an overview of an ATM network to which the inventive system may be applied;

Figure 2 is an overview of a system in accordance with the invention;

Figure 3 shows a roll forward of transactions;

Figure 4 shows a cross connect creation; and

Figure 5 shows a cross connect roll forward.

Figure 1 shows the general architecture of a system to which the invention is applicable. An ATM network manager is managed by a commander 2 implementing the present invention, which communicates with the network manager via a protocol Qs, which is a published interface from Newbridge Networks Corporation.

Roll forward is required on a node after a catastrophe. The major assumptions behind this design are:

1. The customer has a failing switch
2. Returning as many components of the switch to service as fast as possible is by far the most important task.
3. The switch must be in a consistent state at the end of the recovery to reduce requirement for manual intervention.
4. A clear log of failed RollForward transactions must be provided to the customer.

This feature is intended for use by customers when a catastrophic event has caused the EWSX switch to require a reboot from a backup. This feature enables returning the Node to the state it was just before the catastrophe by performing a roll forward of transactions sent to the Node since the last backup.

The ATM-C Roll Forward is a product which supports the Siemens EWSXpress V3 ATM switch and applications. It provides a mechanism for the recovery of the Node from a catastrophe via the roll forward of transactions performed on the Node since a backup was performed.

A central process on the ATMC is receiving messages containing information on all transactions performed on the Node. The feature logs the transactions in separate file for each node. Applications, such as CR, SS7 etc. are using the ATMC API Logging function to send the logging messages to the proper ATMC controlling the Node. It is

assumed that only transactions that are successfully applied to the node are logged. This implies that the logging of transactions must be done by the application only at the end of the transaction. The transaction itself contains a time stamp indicating when it has started. This information is used for the sequence of roll forward.

The Roll Forward of transactions enables the restoration of the Node Database to its state from before a catastrophic failure. Proper recovery of all the data requires synchronization between all the applications controlling the Node. This feature controls the sequence at which transaction that were applied to the node before the crash are being send to the node. This feature reads the log file generated by the logging mechanism.

The Logging and Roll Forward mechanism of the ATMC require that all applications that are acting on the Node use both of the following interfaces:

- Logging interface to log all changes to the Node
- Roll Forward Interface - To enable roll forward of all actions performed on the node.

The logging interface is part of the CKATMCAPI class and contains the following function

```
void ATMCClient :: LogOperations (ATMCAddressType Atmc, // Address of the ATMC Used for logging
        char *NodeDn, // The DN of the Node for that operation
        time_t StartTime, // The time at which the operation started.
        char *Application, // Application Name
        char *Description, // String representation of the action - used for manual roll forward
        unsigned long DataSize, // Size of the data to be logged for the event
        void *Data, // The transaction data
        long Delay = 0, // Specify amount of time in milliseconds to wait before executing next command
        int WaitCompletion = 1 // Indicates if the Roll Forward mechanism should wait for completion of this command
before sending the next one.
        );
```

Application name : Is a string used for identification of the application logging the transaction. This is displayed to the user during the Manual roll forward (See section **Error! Reference source not found.**) and as

	an identifier to the roll forward mechanism of the application performing the transaction during the roll forward.
Description:	A user readable string containing all the information used by the transaction. This information allows the user to screen out transactions during roll forward.
Data:	This is the binary data used for roll forward. This information is send back to the application during roll forward.
DataSize:	The size of the data in bytes.
WaitCompletion:	See below.

During roll forward the application will send roll forward request to all the applications that logged actions.

The following pseudo code illustrates a typical applications written for roll forward:

```
Fd = RollForwardOpen ()
for ever {
        select  on fd and others
        if (message on fd) { // E.G. Roll Forward
                RollForwardGetTransaction
                Process Transaction
                RollForwardSendConfirmation
        } else { // Message from other sources
                something else
    }
}
```

Following are the interfaces for roll forward:

int RollForwardOpen ()

This interface opens a socket connection that accepts messages from the Roll forward Application. The return value is a file descriptor that can be use in a select statement.

int RollForwardHasTransaction ()

This application returns 1 if there is an action waiting for roll forward in the message queue. 0 is returned if there are no messages in the queue. This is not an indication that the roll forward is done see **Error! Reference source not found.Error! Reference source not found.**

int RollForwardGetTransaction (unsigned short &DataSize, void * &Data, int &NeedConfirmation)

This enables the application to get the next action from the roll forward interface. The first parameter is the size of the logged data. The second parameter a void pointer to the logged Data. The third parameter indicates whether the roll forward process is going to wait for a completion message from the application, when it is set to 1 the application must use RollForwardSendConfirmation once it is safe to roll forward the next transaction. This is set to 1 only if the WaitCompletion flag was set to 1 when the application has logged the transaction.

The return value of this command is: 0 - Fail 1 - Data Available 2 - Done with roll forward

void RollForwardSendConfirmation (int Status = 1)

This command must be used if the WaitCompletion Flag was set to 1 in the RollForwardGetAction request. Use this function to indicate to the roll forward mechanism that the application is ready to receive the next transaction. This function is used only by applications that set the WaitCompletion flag in the logging record.

When the Status flag is set to 0 the RollForward mechanism assumes that the transaction has failed and will generate the appropriate log to the customer.

On startup the roll forward application will get the information of the node requiring roll forward, the time at which the last backup has been performed. (This feature assumes that the backup generation was already applied to the node). Then, It will scan the log file and determine the list of applications that were active during the time from the backup.

The roll forward application will attempt to connect to all the applications obtained in the previous step. If any of the application fails to respond the roll forward application prompts the user to start the application manually on the appropriate machine.

As long as not all transactions from the log have been rolled forward the roll forward application will:

- send transactions to the appropriate application
- Wait for the proper reply.

During manual roll forward the RollForward application prompts the user before each transaction is send to the other applications for execution.

As long as not all actions from the log have been rolled forward the RollForward application will:

- Prompt the user for the execution of the next command
- Abort or skip command as per user input or
- send actions to the appropriate application
- Wait for the proper reply.

See Figure 2 above

The Roll Forward application will send a Finish message to all the other applications.

It is assumed that all logged operations were:

Authenticated - That during the initial invocation of the transaction the application has verified that the user have the adequate credentials that enable him to carry on the specific operation.

Successful - During the roll forward of the commands it is also assumed that all commands logged through the logging mechanism were successfully applied to the Switch.

Error recovery during roll forward is via the existing transaction roll backward of the applications. Transaction roll backward is designed as part of standard transaction handling. During roll forward the built in Roll Backward of transactions should work as during normal operation. Since roll forward is done for transactions that were successfully applied to the node, it is anticipated that this mechanism will rarely be invoked. However, in the case of hardware failure, during roll forward, of one of the modules on the Node a transaction may abort. In this case the transaction should behave the same way as it would under normal conditions - that is roll backward. This will leave the Switch in the most consistent way possible. Logs indicating the failure will appear in a standard log. Since we do not necessarily have a GUI connected to the applications the SendConfirmation message to the RollForward mechanism should indicate the failure so that the Roll Forward mechanism can generate additional required logs.

It is known that this mechanism not manage to completely restore the Node to its original condition under the following conditions:

1. A race condition between two applications or transactions while setting an attribute for the same object. In this case there is no way to determine which application will be the last to set the attribute, and therefor to assure proper operation.
2. There is a physical problem with one of the components on the Switch.

The following applications are known to be using the logging and roll forward API stubs at the release 2 time frame:

- ATMC base.
- Call Routing

- SS7
- Q3PS
- Autocraft

The following example is an example of creation of a cross connect. This type of operation is a very typical one on the ATMC and at any given time we might have multiple cross connect transactions in operation.

As can be seen from the above figures, using cross Q3 roll forward will result in reversing the state of the two object. That is the connection represented by Object 1 will be inactive while the connection represented by Object 2 is going to be active. This is in contrast to the original state of the connections where the connection represented by Object 1 was active and the connection represented by object 2 was inactive.

Q3 logging at the object level is the original proposal prepared for CR/SS7. In this design the application is logging the Q3 objects just before they send the command to the Node. Using Vertel this implies that before sending commands to the node the application can translate the Vertel object to a BER representation using the Vertel "Ber_Encode" template then send the information to the logging mechanism using a function as follows:

```
void ATMCClient :: LogOperations (ATMCAddressType Atmc, // Address of the ATMC Used for logging
        char *NodeDn, // The DN of the Node for that operation
        time_t StartTime, // The time at which the operation started.
        char *Application, // Application Name
        char *Description, // String representation of the action - used for manual roll forward
        long DataSize, // Size of the data to be logged for the event
        void *Data, // The BER of the transaction object
        long Delay = 0, // Specify amount of time in milliseconds to wait before executing next command
        int WaitCompletion = 1 // Indicates if the Roll Forward mechanism should wait for completion of this command
before sending the next one.
        );
```

During the roll forward of the transaction the Roll Forward agent will perform similar tasks to the ones described above. The only difference between is that instead of

sending the Q3 actions to the applications for execution the BER data of the Q3 actions is converted to a Vertel Object representing the Q3 command (Using features of the Vertel Stack T.B.D) and the action is than sent to the Node for execution. Q3 is a standards based interface for managing network elements. The Siemens EWSX V3 is managed via a Q3 interface.

Logging at the Socket level is similar to the design described in the previous section, except that instead of the application calling a special function for the purpose of logging of the commands the data is intercepted and logged at the MP120 Daemon. This design has the following problems:

- Requires modification of the MP120 (Vertel).
- Does not support any other stack such as the one used by Q3PS.
- Selection of 'actions' during the roll forward is very hard since there is no information as for the generating application and user (Official requirement).

THE EMBODIMENTS OF THE INVENTION IN WHICH AN EXCLUSIVE PROPERTY OR PRIVILEGE IS CLAIMED ARE DEFINED AS FOLLOWS:

1. A method of monitoring transactions in a network application, wherein a transaction log is maintained at a node, a separate log is maintained, a playback is implemented which relies on the application to replay the original knowledge context of the transaction, and if this replayed mechanism fails, a transaction rollback mechanism is invoked to restore both the application and the node to a mutually consistent state.

2. A method as claimed in claim 1, wherein a centralized log is maintained similar to the log at the nodes, and when a playback is implemented the log is played back in a synchronized manner to ensure node and application consistency.

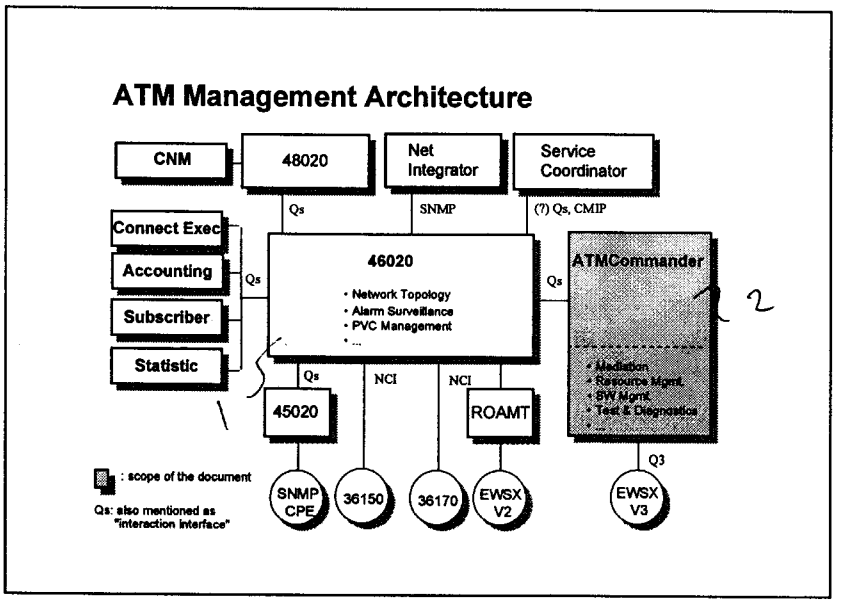
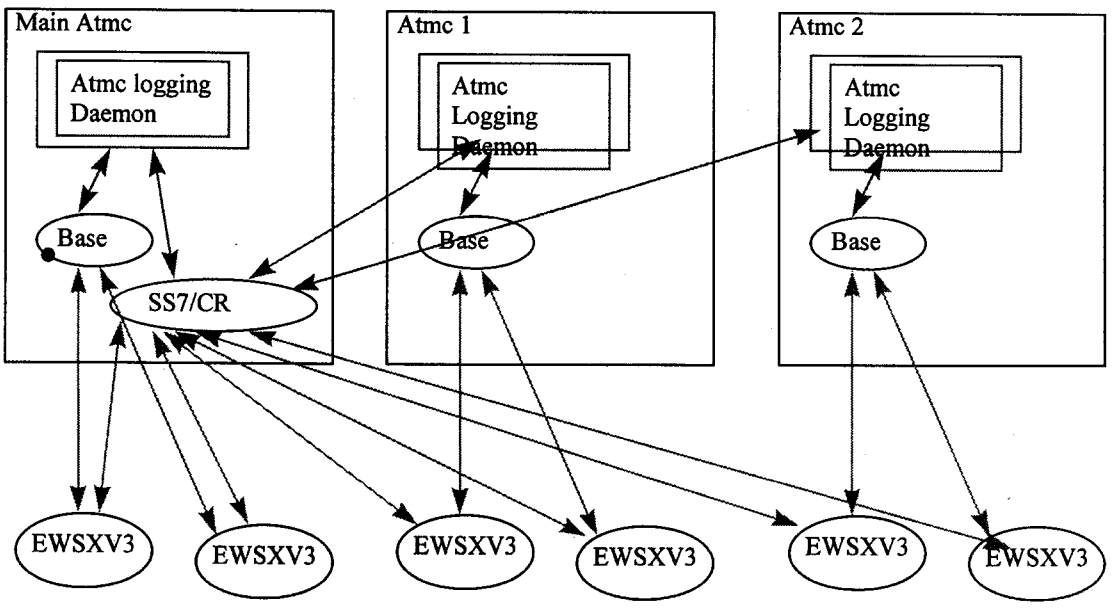


Figure 1



The Atmc Logging Daemon
1. Log transaction.
2. Play Back Transaction
3. Backup Switch
4. Restore Switch

Data Required For Logging and Playback
•Switch Address
•Time
•Application
•User Id
•Description
•Data

Figure 2

Marks & Clerk

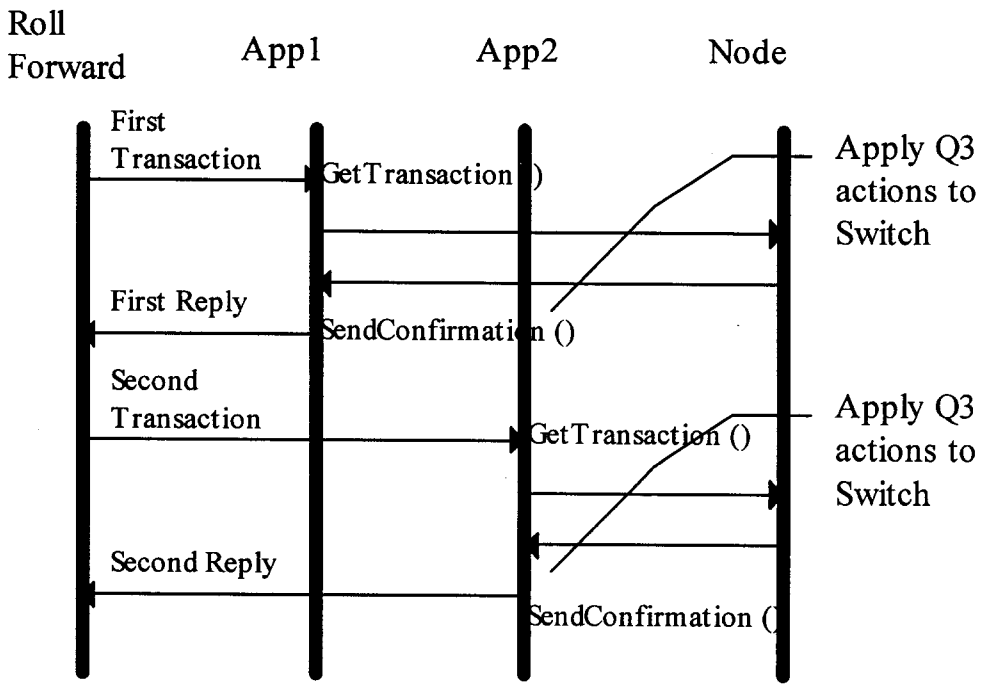


Figure 3

Cross Connect Object can be created in an inactive state and than it's state can be modified to enabled. When creating the xcon you specify the two VPIs and send the request to the ATMC fabric (1 and 2). The reply from the ATMC fabric (1r and 2r) contains the DN of the xcon. If you send two requests there is no guarantee as for which is going to be processed first so the second (2) may be executed before the first as demonstrated in the following figure resulting in request 1 creating object 2 and request 2 creating object 1.
The objects are created inactive and I will assume that transaction A will activeate it's cross connect (Object 2) and transaction B will not.

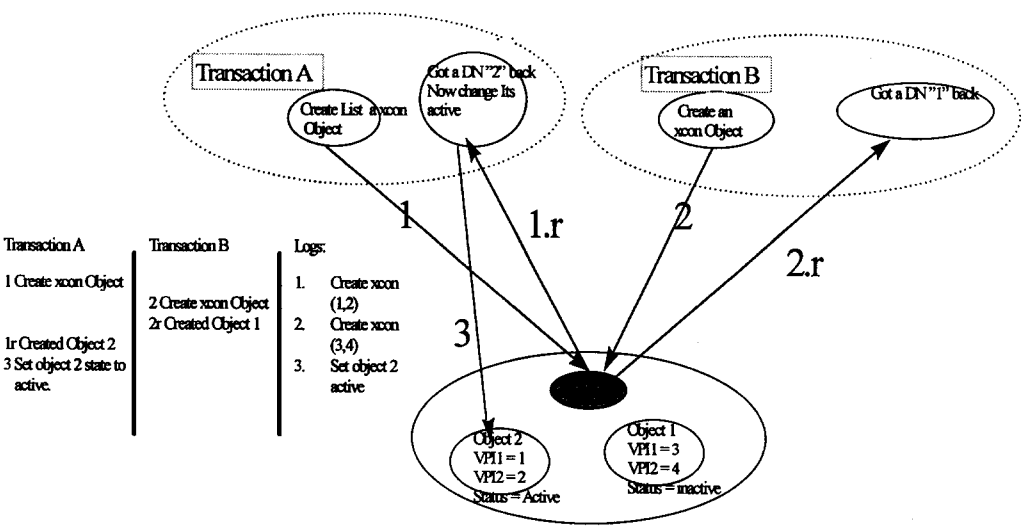


Figure 4

Marks & Clerk

During Roll forward we create both objects in the order recorded in the log. The result is that we create the object for vpi1 and vpi2 first resulting in creation of object 1 and only then we create object 2. The next transaction in the log is to modify object 2 state to active resulting in a swap in the object states.

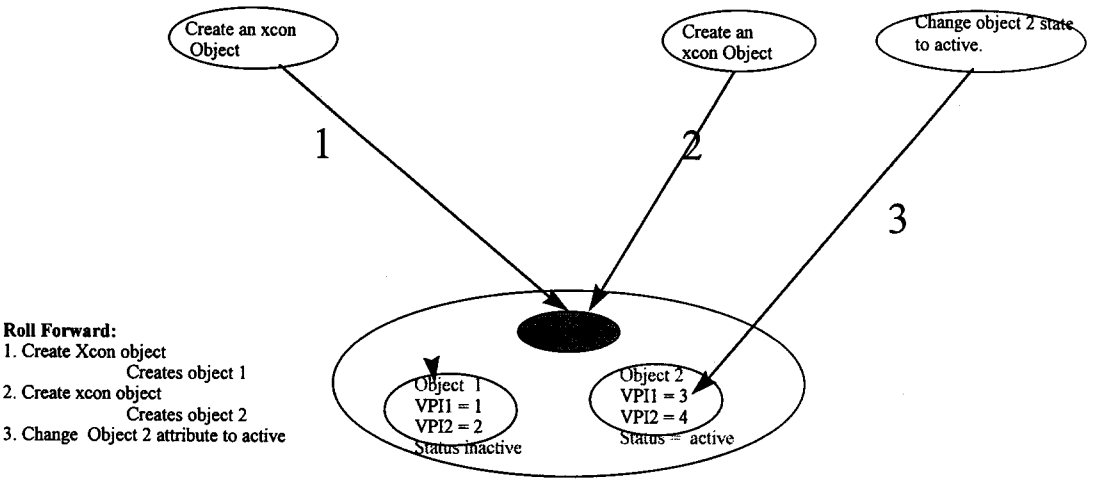
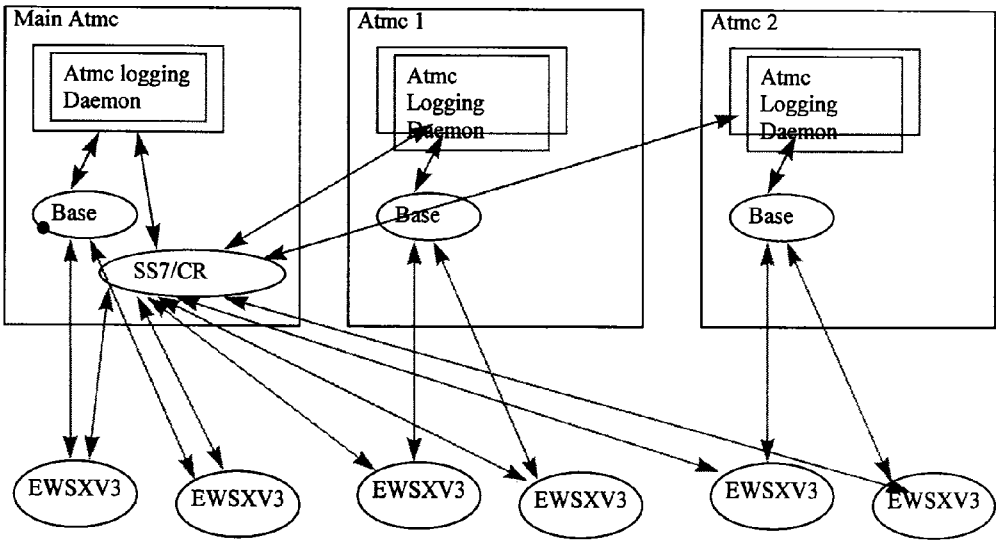


Figure 5

Marks & Clerk



The Atmc Logging Daemon
1. Log transaction.
2. Play Back Transaction
3. Backup Switch
4. Restore Switch

Data Required For Logging and Playback
- Switch Address
- Time
- Application
- User Id
- Description
- Data