BROWN UNIVERSITY
Department of Computer Science
Master's Thesis
CS-93-M3

# ''Interactions Recovery System (IRS): Rollback and Recovery for Multidatabases in a Heterogeneous, Distributed Computing Environment''

by

Robert T. Baynes

# Interactions Recovery System (IRS): Rollback and Recovery for Multidatabases in a Heterogeneous, Distributed Computing Environment

by

Robert T. Baynes
B.S., Jacksonville University (Fla.), 1979

Submitted in partial fulfillment of the requirements for the Degree of Master of Science in the Department of Computer Science at Brown University.

March, 1993

This research project by Robert T. Baynes is accepted in its present form by the Department of Computer Science at Brown University in partial fulfillment of the requirements for the Degree of Master of Science.

Date 4/8/93

Stanley B. Zdonik

Stanley B. Zdonik

# Interactions Recovery System (IRS): Rollback and Recovery for Multidatabases in a Heterogeneous, Distributed Computing Environment

*Robert T. Baynes*

*7 March 1993*

# Interactions Recovery System (IRS):

# Rollback and Recovery for Multidatabases in a Heterogeneous, Distributed Computing Environment

Robert T. Baynes
Brown University
Providence, Rhode Island 02912

Department of Computer Science

7 March 1993

*Abstract*

*This document presents the Interactions' Recovery System that is part of the Interactions MultiDatabase Project. The document presents a discussion of the rollback and recovery methods implemented for the project. Rollback and recovery is a well understood area of study for the general database system. However, when applying the paradigm to a multidatabase system rollback and recovery presents some challenges in maintaining both consistency across the multidatabase system and the local database system. This project attempts to make this application easier.*

# 1 Introduction

The multidatabase architecture provides a system that allows for a number of heterogeneous, distributed databases to be utilized for one application. The classic example used is that of a travel agent [NOD91]. The travel agent is required to plan a trip that uses various, but different, reservations databases (e.g. hotel, car rental, airline, etc.). The travel agent can either access each database individually, or in the case of Interactions, can create one task and have the system handle the *interaction* with each database.

The Interactions system design is treated like any other database system. Transaction consistency and recoverability must be addressed. While issues such as serializability are well understood, the issue of recoverability is not as straight forward. What complicates this area of design is the desire to optimize the recoverability of interactions to speed up the execution of recovery by discarding items that do not have to be recovered, while maintaining database integrity.

## 1.1 Multidatabases present unique rollback and recovery problems.

There are a number of issues that can be applied to multidatabase systems that are not necessarily required in less complex systems. For example, rollback and recovery of single site databases is straight forward and does not require knowledge or concern of a network. This is quite true for single user database systems (such as those based on personal computers) and those based on a centralized systems (such as large mainframes). This means that when a transaction is required to be rolled back or recovered from some previous state, the database does not have to consider the distributed environment of a multidatabase. Rather, it is "allowed" to ignore the network, even if the user is on some type of network for access and therefore there are no independent actions that require the database to address.

On the other hand distributed database systems must take heed of the issue of network connectivity and availability. The distributed system must have a mechanism to rollback and recover transactions without regard to the fact that it is distributed. That is to say, even though the topology of the system is a distributed topology, the database system must be designed in such a way that when a failure or other action requires rollback or recovery of the database the system can handle the myriad of network problems that may be introduced. To put it succinctly, the rollback and recovery system of a distributed database requires the ability of that system to be able to deal with network problems, e.g. partitioning, intermittent failures, communication delays, etc., while insuring database integrity and consistency.

While the distributed nature requires one to be especially mindful of rollback and recovery of multidatabases, heterogeneity is not an issue in the present context. The distributed multi-database, Interactions, is homogeneous across the nodes of distribution. Each database on a specific node may (and in all likelihood will) be different; the superset architecture of the distributed database that accesses all local databases is homogenous. This allows the designer to not be required to have knowledge of specific database recovery mechanisms when designing and implementing and rollback and recovery system for a multidatabase system such as Interactions. It does not preclude the need for knowledge of the local database system when actually doing an undo of some transaction at the local level, but for the design of the distributed system that knowledge is secondary.

Host autonomy is maintained throughout. The local databases do not actively cooperate to effect multidatabase consistency and recovery, Mongrel provides this support. Finally, host heterogeneity is an issue. Due to the potential large number of possible hosts and operating system varieties on a network. Interactions' Recovery System needs to be aware of these differences and, if at all possible, take advantage of them.

## 1.2 Interactions.

Interactions is a global transaction model that supports access to a multidatabase system [NOD91]. The purpose of Interactions is to provide an environment to define and execute one task that spans more than one database in a multidatabase system. The Interaction structure includes: **steps, actions, events, and strong and weak conflicts.**

An Interaction consists of a partial ordered list of global transactions. Each global transaction is a set of global subtransactions each of which executes on a single local database.

A **step** is a complete set of instructions (or operations) for a single database. A step cannot contain operations for other databases or have unrelated operations for another step but the same database.

An **action** is a partial order of steps.

**Events** are those occurrences of database operations outside the realm of the particular Interaction we are interested in. An event can take place from another interaction or a local database transaction. Events are important as they can affect the operation of the interaction of interest, by accessing the same areas of the database.

A **strong conflict** is a definition of an operation that cannot be intermixed with the Interaction. Strong conflicts are enforced within individual atomic transactions on the local databases. When another transaction attempts to execute some strongly connflicting operation, that is prevented by the local database's transaction manager (because it enforces serializability).

A **weak conflict** is defined as one that preserves the conditions in a database for a specific span of an Interaction. It also provides what is required to recover if a weak conflict is violated [NOD91].

A **task** is a unit of multidatabase work that consists of one or more local database transactions. An example of a task is the ubiquitous travel agent example [NOD92]. Each local database transaction consists of steps and each step is a single executable entity on a local database. Each step is defined in a local database step library. The step is translated into local database instructions and then passed to the local database for execution.

Task recovery is the action taken by the multidatabase system to return the local database systems back to a known state of consistency after some failure or exception and then proceeding onward with the task at the point of recoverability. Task recovery does not need to go forward but can end the task after reinstating local database integrity.

The multidatabase access assumptions include: non-atomic tasks, procedural tasks, cooperative tasks, and compensation-based recoverable tasks. It is the recovery of tasks that the IRS is most interested.

## 1.3 Multidatabases.

The multidatabase system has two levels. The local level is that level that consists of all the local databases and all of the those actions that would be contained at this level. At the local level one would observe steps being converted to local database instructions, and those instructions being executed on the local database.

The global level is the level where the Interaction Manager interfaces with the Multidatabase application (via Interactions) and the associated Agents (local level) that are assigned to the local database. At the global level, Interactions are defined, global transactions are created with subtransactions being dispatched to various local databases.

For Interactions our multidatabase integration strategy includes:

**Heterogeneity:**

It is assumed that the local databases support: serializable, ACID transactions; no other assumptions of heterogeneity are made.

**Uniform Access:**

There is a global interface provided to the multidatabase applications that effectively hides the local database manipulation language. This is accomplished by defining "step libraries". These libraries are defined for each local database and contain step to local database instruction mapping. Additionally, the processes of Interactions themselves enforce uniform access through standard interfaces. By applying the paradigms of Object Oriented programming for data privacy and access of data via methods the processes maintain the uniform access strategy.

**Local Database Autonomy:**

**Design Autonomy**: The assumption is that the local database's transactions are atomic, serializable and recoverable.

**Communication Autonomy:** The local databases cannot communicate with each other. The communications of the Interaction Manager are to each local database and provide no communications path that the local databases can employ to communicate with each other.

**Execution Autonomy:** For execution of transactions the local databases are assumed to:

1. Execute their transactions in any way, not being dictated by Interactions.

2. Execute non-multidatabase transactions (that is local transactions) initiated outside of the multidatabase architecture under the local database constraints. Interactions does not interfere with these transactions.

There are no assumptions made about the distribution of the local databases; they can be distributed in anyway.

## 1.4 Format of the remainder of the paper.

Section 2 will discuss some of the literature on log based rollback and recovery. The discussion is not exhaustive.

Section 3 will cover the implementation issues and discuss the design of the IRS.

Section 4 is a discussion of the operation of IRS.

Section 5 will provide several scenarios for IRS.

Section 6 will summarize the work.

## 2 Log based Rollback and Recovery.

### 2.1 Review of the literature.

Traditional Recovery Systems are based on utilizing some log of changes to insure consistent databases when recovering from a failure [BHG87]. The storing of information for a committed database occurs on stable store, but separate from the database. If a recovery is necessary the recovery manager must be able to resolve the information in the stable store to restore the committed database to a known state before recovery.

The recovery manager must be able to handle both the rollback (undo) of information and the re-execution (redo) of certain transactions in order to restore the database. In the current study we under take to establish a system that will be able to meet all of these needs.

Distributed database systems require a more extensive log and a system that is able to communicate to many nodes. The issue of distribution is determining how to maintain a consistent global state across distributed resources [LI91]. As [LI91] notes there is no clear algorithm for maintenance of this global state. It is proposed by [LI91] that optimistic checkpointing schemes are not required and even produce negative effects at the expense of the actual application.

### 2.2 Checkpointing.

While the argument bears merit in a tightly coupled multiprocessor system, for Interactions we look at a loosely coupled, geographically distributed system. It is important to maintain an optimistic checkpointing and recovery scheme. Optimistic checkpointing allows Interactions to be confident that at anytime during the processing of any interaction a call to the recovery system will produce the correct results. Without the optimistic system in place we do not have that same confidence. Further, given the distributed, loosely coupled nature of Interactions, recovery across a wide area network becomes extremely problematic without the optimistic viewpoint.

This requires that the logs created and maintained by IRS be extensive and each node have its own autonomous log and daemon for logging (this goes against [LI91]). In IRS we create a fully distributed system in that the Interactions Manager (IM) has a logger daemon (Interaction Logger Daemon (ILD)) at its host. Each local database (or agent in Interactions parlance) has its own logger daemon and log. (See [MOH91] for detailed discussions on these points).

The use of checkpoints to support a recovery system has been well documented ([KOO87], [GOL91], [LONG91], [CRI91]). In our current application checkpoints present an important "fail safe" feature of IRS. By maintaining checkpoints (in this work we utilize naive checkpoints [KOO87]) and logs in parallel the system presents to Interactions a set of data points to maintain a consistent set of databases. In fact, this design allows for a complete network failure between an agent and the IM and the IRS would be able to insure that the agent's database will be consistent with its known state prior to the network failure.

The checkpoints also provide for future expansion of the Interactions system to used enhanced recovery. Enhanced recovery means the ability to maintain some in-between state of a transaction after the recovery process. In this way, Interactions would no longer have to make an all or nothing decision on a transaction.

### 3.1 The application of write-ahead techniques for the log system of a multidatabase.

The write ahead protocol allows for the log to always be guaranteed to be more accurate then the stable store database [MOH91]. Write ahead writes the actions of the database to the log before committing them to the stable store database. The Write ahead protocol does not require all of the databases to enter the final vote (of a 2 PC protocol) for a commit. Therefore each local log reflects a current state of the database that is better known then the Interactions database.

However, in Interactions we violate the write-ahead protocol. While it can be applied somewhat, Interactions is not aware of the various commit protocols for each of the potential database systems that it may interact with. Therefore, a straight write-ahead would not be appropriate.

The write ahead techniques that we apply are used to maintain the checkpoint files. Commits of global transactions go to the ILD, but all of the necessary information for any type of recovery is available in the check point files.

Our write ahead protocol enhances the checkpointing mechanisms implemented for multidatabases as the log entries after a checkpoint are used to restore the database to a higher state of consistency. But the checkpoint is used in conjunction with the log entries to further validate the database integrity and present the most current information possible in recovery.

# 3 Implementation of the Log Based IRS.

IRS utilizes a distributed log based schema for rollback and recovery. This type of architecture allows for the system to be fully distributable and adaptable to additional local databases as Interactions develop and expand. The components of this logged based system include:

1. Logger servers and clients: The servers and clients are used for all reading, writing and formatting of log entries.

2. Logs: Logs maintain records for rollback and recovery operations. There are logs for all local database systems and one log for the Interactions Manager.

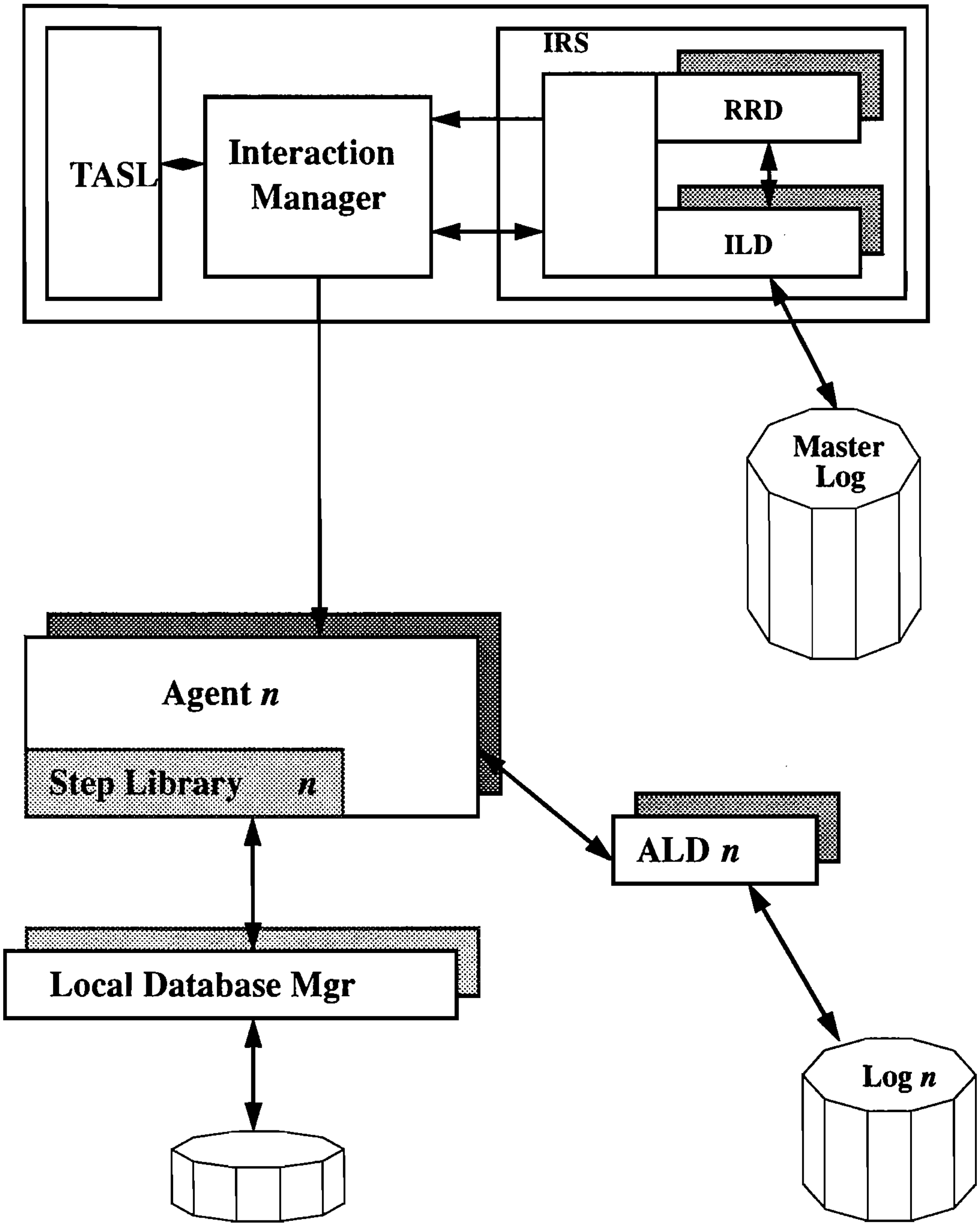
3. Checkpoint Directories and Files: The checkpoint directories and files provide for determining the state of the system during operations and during catastrophic failure recovery. The combination of log records and checkpoints give IRS the information needed to abort, rollback, and redo transactions while still maintaining database integrity.

The IRS writes log records out to a separate stable store environment (a log). These records contain the necessary information for the recovery manager to determine what needs to be undone, redone or discarded.

The following sections develop these components and explain the operations of each in greater detail.

## 3.1 The Interaction's Recovery System Architecture.

The Interaction Recovery System fits into the multidatabase architecture as shown in Figure 3-1.



*Interface Diagram for the Interaction's Recovery System*

*FIGURE 3-1*

There are two components of the global rollback and recovery utility (IRS). The Interactions Manager (IM) will make a call to the IRS when it is necessary to either log an action or rollback/recover[1] some action. When the IM makes the call the interface is to a "frontend" process that sets the correct format and calls the Interactive Logger Daemon (ILD). This utility is resident on the Interactions host and consists of:

**Interaction Logger Daemon (ILD):** A daemon that executes on the Interactions host. This daemon maintains all log entries to the Interactions Log. In addition, it maintains the checkpoint directory structure for the global log. It maintains interfaces to the RRD for recovery.

**Rollback and Recovery Daemon (RRD):** A daemon that executes on the Interactions host. This daemon controls all rollback and recovery processes. It is the process that evaluates and initiates the compensation and replacement algorithms for rollback and redo.

While the IRS is a global logger system, there is a subcomponent of the logging system to maintain the local database transaction that the Interactions system executes. This is at the agent (or local) level. this component is the **Agent Logger Daemon(ALD)**. This is the daemon that executes on the Agent's (local database) host. It maintains the Agent Log and the files within the global transaction directory during execution of the local database transactions, in much the same manner as the ILD.

As noted, this system has a two-level log. The logs are resident on the host that the respective logger daemon is executing.

Interactions Log: This log is resident on the Interactions host and is the global log for the Interactions Manager. It is updated by the ILD. The purpose of this log is to record all global information and provide the necessary information to the RRD for rollback and recovery.

Agent Log: This log is resident on the Agent host and is the local log for the Agent and Local Database. It is updated by the ALD. The purpose of this log is to record all local information and provide semantic undo information required to properly rollback and recover transactions on the local database. There are as many agent logs as there are local databases. The current work does not cover the details of the agent log design or implementation. The concept and basic underlying theory is parallel to the ILD operation and design.

[1] When rollback/recovery is mentioned in this paper, it means the actions of either aborting, redoing or rolling back a global transaction. A global transaction is always the unit of rollback/recovery.

Compensation-based recovery will be used in Interactions. Further, optimized compensation-based recovery will use the Replacement Recovery Algorithm [NOD91-2]. These algorithms will be implemented and executed in the RRD.

The Interaction's Recovery System needs to be distributed to take advantage of the architecture of Interactions and of the distributed network environment that Interactions runs under. Rollback and recovery information needs to be located at the node on which the recovery needs to take place (i.e. the node that executed the original subtransaction).

The Interaction host provides a global Interaction's Recovery System that encompasses the following:

1. Methods to access the local database logs (the agent log) for compensating step recovery of transactions.

2. A general log of Interactions (the Interactions Log).

3. Methods to perform the compensation-based recovery on a complete Interaction or parts of an Interaction.

Rollback and recovery will utilize semantic undo (compensation) to maintain a consistent database (both local and global databases). Semantic undo information must be provided by the agent for the local database to the Interaction Logger for later use. The actual undo information for local databases will be maintained in the local Agent Log.

Rollback or recovery can be initiated by either Interactions or anyone of the Interaction Agents. Recovery can be initiated locally with the Interaction Manager providing Agent requested information from the Interaction Log. However, a more typical action is for the Interaction Manager to initiate the recovery of a global transaction, at the request of the specifier of that Interaction.

Checkpointing is supported in Interactions. While this places an added layer on the logging utilities, it is used to guarantee that a database, requiring a rollback and recovery has its integrity maintained.

## 3.2 The Architecture of the Interaction and Agent Loggers.

The Interaction Manager is the interface to the Interaction Logger for all Interaction task logging. The ILD is the heart of the rollback and recovery mechanisms for Interactions. The Interaction Logger is a daemon that is called via an RPC when required to accomplish some task. The Interaction Logger is implemented on the Interactions host and all local database hosts that have an Agent running.

The Interaction Logger's functions include the following:

1. Verify all information for logging contains the information required to accomplish to rollback and recovery of a database.

2. Log all information in the Interaction Log, giving writes to the log priority over reads.

3. Provide information (in the correct order) to the RRD during rollback and recovery. During this time the Interaction Logger gives reads priority over writes.

The Interaction Agents interface with both the Agent Logger and the Interaction Manager to record needed information about tasks and transactions. Section 4.2 details the files associated with the IRS. The following is a general description of the overall architecture.

During rollback and recovery phases the Interaction Logger does a single pass through the log (that is, the Interaction Logger does not do random reads to obtain rollback and recovery information).

Every instance of a logging action is flushed to stable store, either to the checkpoint directory, or to the log, immediately after each action. The Interaction Manager has the ability to direct the ILD to write items directly to the log.This by passes the checkpointing scheme but, the ILD does verify instructions from the IM by reviewing the check pointing directories and initiating the correct action to maintain its consistency. Therefore, even though the IM can "circumvent" the intended design, the ILD insures its consistency throughout the logging process.

The ILD reads a state and execution order file to determine the order and dependency of each transaction when a rollback/recovery is initiated. This file contains information on the order of a particular transaction to its peers within the Interaction. An execution order defines where the transaction falls within the execution stream of the Interaction. State order is the definition of a transaction'sorder based on the state of another transaction. Both state and execution order must be analized to properly recover transactions. This state and execution order information is provided to the ILD from the IM. The IM receives its information from the command interpreter (TASL)[NOD91]. The ILD will pass host and compensation log record information (typically log record numbers) that the RRD will use to undo database information at the Interactions or Agents level. The SE order information is used by both the ILD (to construct the correct order of Global Transactions) and the RRD (for use during the compensation/replacement process of rollback and recovery).

### 3.3 The Interfaces.

The Interaction Manager is the main interface agent with the ILD. It will provide the following information to the ILD:

1. General information of all Interaction.

2. Read/write information for each committed Global Transaction of the Interaction, provided by the Agents, via the IM.

3. A delimiter/date-time stamp for entry into the log.

4. Rollback and recovery requests for Interactions and the local databases.

The ILD will maintain the following interfaces:

1. Acknowledgment of information received from the Interactions Manager.

2. Read and Write access to the log. The read and write access priorities change depending on the required actions.

3. The ILD will provide the log records necessary to undo Interaction tasks, local database interactions and dependency trees as rollback and recovery execute.

The Local Agent will interface with the ALD and provide the following information:

1. Semantic undo records for possible rollback and recovery.

2. Log Record numbers associated with each semantic undo record in the ALD.

3. A delimiter/date-time stamp for each entry into the log.

The ALD will not interface directly with any other Logger Daemon (neither the ILD nor another ALD). The ALD will only maintain the following interfaces:

1. An interface with the Local Agent to provide information requested by the Local Agent or the Interaction Manager. The information provided will be either the actual undo information to execute the undo or record information used for the ILD process.

2. Read and Write access to the log. The read and write access priorities change depending on the required actions.

3. A delimiter/date-time stamp for each entry into the log.

# 4 The Operation of the Logger.

## 4.1 Initiation of the Logger Daemons.

The ILD is initiated at boot time for the Interactions Manager as a daemon process that is awakened by RPC calls. The records written to the log at the start of the ILD will be the following:

DELIMITER RECORD.

ERROR/EXCEPTION RECORD: This record will contain the initiation message (considered an exception) from the Interactions Manager.

The ALD is initiated by its Agent at boot time of the Agent. The records written to the log at the start of the ALD will be the following:

DELIMITER RECORD

ERROR/EXCEPTION RECORD: This record will contain the initiation message (an exception) from the Interactions Manager.

Both the ILD and ALD are initiated as daemon processes of the IRS (for the ILD) and the Agent process (for the ALD). The processes is awakened when required to perform some task by an RPC from the IM or agent.

Error checking takes place AFTER the initiation of the logger daemons to prevent confusing initiation problems that might occur prior to log attempts. No logging will take place until after all error checking has been completed. If errors are discovered the respective logger (ILD or ALD) logs the error in an ERROR/EXCEPTION record and returns an error to its caller.

## 4.2 The active files of the Logger Daemons.

The Logger File System is set up to provide both a consistent method for all databases to maintain current operations and an easy interface for the logger to receive the necessary information for logging in the Multidatabase logs[2]. **Figure 4-1** is an illustration of the file system structure.

[2] The reader is reminded that there are two levels of logs: 1)the master log written to by the ILD, and 2) the agent log written to by the ALDS.

# IRS Checkpoint Filesystem Structure

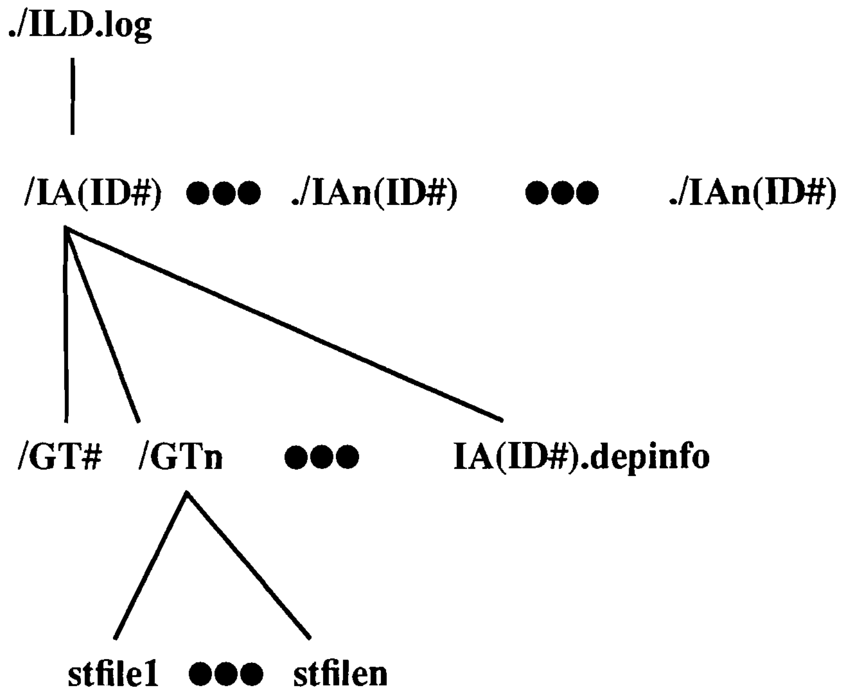

./ILD.log
/IA(ID#) ●●● ./IAn(ID#) ●●● ./IAn(ID#)
/GT# /GTn ●●● IA(ID#).depinfo
stfile1 ●●● stfilen

*FIGURE 4-1*

This structure is created at the IRS initialization time and maintained throughout the logger lifetime. The log daemon will execute and create the structure as follows:

**ILD.log (or agent*n*.log):** This is the actual log kept of all Interactions, transactions and subtransactions. It is always re-created at boot time of the log daemon. The log daemon will append to the existing log, or if there is no log the daemon will create a new log.

**/IA.(ID#)**: This directory is a specific temporary directory for each Interaction. The directory is created at the time the Interaction commences. It is removed after the final Global Transaction has committed and the information has been written to the log.

**/GTn**: This is a directory for a particular Global Transaction. It is created at the time the Global Transaction starts executing. In this directory are files that designate each subtransaction (stn). Each subtransaction file contains a log of all operations that have taken place up to the time one might examine the file. As the subtransaction is completed and written to the log, the file is then deleted. When all of the subtransactions have completed the Global Transaction directory is deleted.

**./IA(ID#).depinfo**: This is the file that contains all of the state and execution order (SE order) information for the specific Interaction. It resides at the Interaction directory level and is deleted when the Interaction is committed to the ILD log. This file is written to the log for future use during recovery.

The log daemon utilizes this type of file system structure for a number of reasons.

By using directories for the major transactions of an Interaction the multidatabase system is given the flexibility to have the Global Transactions on different compute nodes. For instance, if a particular Interaction had two Global Transactions executing, each Global Transaction log directory could be resident on the actual database node that the Global Transaction starts processing from. Figure Three is an illustration of this distributed property.

The structure presented provides an easy system to troubleshoot in case of failures. This becomes very important if a particular node fails in the middle of a large Interaction. The programmer can traverse the Logger file system tracing the progress of the Interaction through the existence of the various subdirectories and files present. In addition, the files for each subtransaction contain ASCII information that provides a trace of the progress of the subtransaction.

The multidatabase concept is based on many distinct databases dispersed throughout a network. The ILD maintains control over the master multidatabase log. It accepts input from various agents via the Interaction Manager and logs the necessary messages. The ALD, on the other hand, is concerned with the specific local issues of the Global Transaction and all of the subtransactions involved with it. Because the majority of the information is local it is only logical that a record of that information be kept at the client site.

The development of a significant portion of code to support such a structure is not required. The development assumes that some Server/Client architecture is in place at the time of the Logger initialization. Because we are dealing with a distributed database environment this assumption is well within reason. At the time of the building (compilation and installation) of the Logger system, various local parameters will be set to insure that the appropriate pathnames are created and the code is compiled with this information (see the Logger design section for more details). In this way, local database administrators can move the various directories to the correct nodes. It also allows the notion that a database administrator would not be required to distribute the directories as previously discussed; they could all remain on one system.

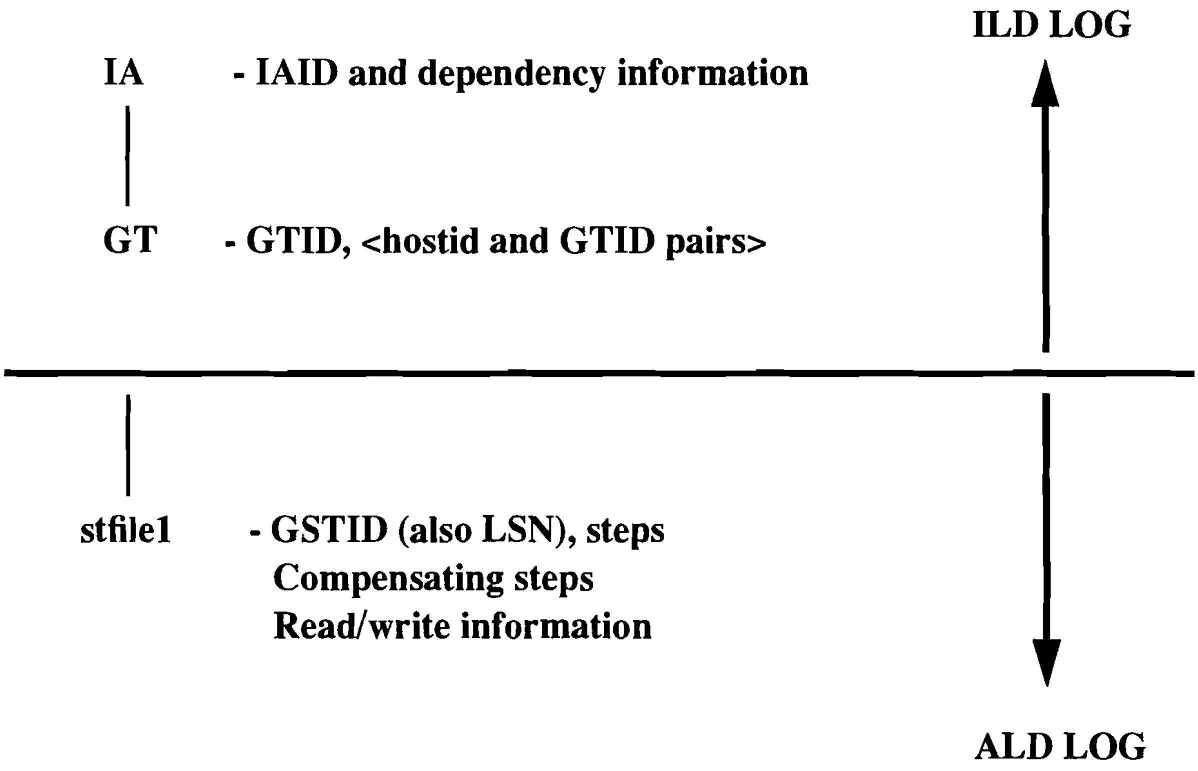
One system is not required to arbitrate I/O traffic from various distributed databases to one central logging system. Rather, each system supports its local logging requirements and only when an actual write to the inter.log file takes place is the Server node required to arbitrate I/O. This load will be significantly less than with a non-distributed logging system.

In a non-distributed logging system the server node is required to do a number of jobs. First, it is required to maintain communications to all of its clients, thereby increasing communication processing overhead. Second, it must spend time arbitrating several accesses to the one log from several different sites. This is not communications intensive, the communication has already been received, rather it is I/O intensive because of the arbitration required between reads and writes to the log. Finally, processing time is required to handle the processing of the log during rollback and recovery phases. It must be able to decipher the type of record, the destination of the record and the contents of the record.

## 4.3 The log format.

The ILD writes various records to the log during the duration of its life. All records are variable length. The fields are delimited by a space in order to ease parsing of the log during Rollback and Recovery Operations.(The records are easily parsed because of the use of C++ I/O functions). The following are the record types and formats that are in the ILD log. Figure 4-2 provides a view of what is logged and at what level.

## IRS Log information



IA - IAID and dependency information

GT - GTID, <hostid and GTID pairs>

stfile1 - GSTID (also LSN), steps
Compensating steps
Read/write information

*FIGURE 4-2*

**ERROR/EXCEPTION RECORD:** This record is maintained for error detection and resolution. The following fields are written by the ILD:

1. **FIELD 1**: TIME STAMP and RECORD TYPE (E).

2. **FIELD 2**: HOST ID (of the host raising the error or exception).

3. **FIELD 3**: ERROR CODE and TEXT INFORMATION.

**DELIMTER RECORD:** This record provides for sectioning of the ILD log. There is only one field in this record:

1. **FIELD 1**: RECORD TYPE (D) and TIME STAMP.

**TRANSACTION RECORD:** This record keeps track of all transactions that occur. The following fields are maintained:

1. **FIELD 1**: RECORD TYPE and TIME STAMP.

   GT: Global Transaction ID

   GST: Global Subtransaction ID

2. **FIELD 2**: HOST ID (of the host associated with the transaction).

3. **FIELD 3**: TRN (A Unique global record number PLUS Compensation log record number from local host)

**INTERACTION:** This is the Interaction that is a complete set of all transactions that are to be accomplished.

1. **FIELD 1**: RECORD TYPE (IT) and TIME STAMP.

2. **FIELD 2**: TRN.

3. **FIELD 3**: Transaction information:

   {All global transactions associated with this task}

The ALD writes various records to the log during the duration of its life. All records are variable length. The fields are delimited by a special character in order to ease parsing of the log during Rollback and Recovery Operations. The ALD contains similar transaction information as the ILD Log, but for every transaction that it logs the ALD log record also contains the semantic undo information required to return the local database to a consistent state during a rollback and recovery operation.

Figures 4-3 and 4-4 illustrate the log record format for both the ILD and ALD.

# InterActions Master Log

```
IA#; TIMESTAMP; BEGIN             At IA Begin
IA#; TIMESTAMP; {GT#, GT#, GT#, . . . }
IA#; COMMIT                       At 2PC START


IA#GT#; TIMESTAMP; BEGIN
IA#GT#; TIMESTAMP; <{GST#:HOST:r/w INFO}; DEP. INF
IA#GT#; TIMESTAMP; COMMIT --or--  At 2PC Commit
IA#GT#; TIMESTAMP; ABORT/REDO



IA#; TIMESTAMP; EXCEPTION CODE; ROLLBACK GT#
IA#; TIMESTAMP; EXCEPTION CODE; REVERSE GT# COMP.
IA#; TIMESTAMP; EXCEPTION CODE; GT# SENT TO IM


IA#; TIMESTAMP; ERROR CODE; TEXT OF REAL PROBLEM
IA#; TIMESTAMP; EXCEPTION CODE; ABORT GT#
IA#; TIMESTAMP; EXCEPTION CODE; SYSTEM INIT
IA#; TIMESTAMP; EXCEPTION CODE; SYSTEM TEST OK
IA#; TIMESTAMP; EXCEPTION CODE; SYSTEM INIT COMP.
```

*FIGURE 4-3*

# InterActions Agent Log

```
GST#; TIMESTAMP; BEGIN
GST#; TIMESTAMP; <{CST1, r/w}, {CST2, r/w}...>
GST#; TIMESTAMP; COMMIT --or--    After 2PC Commit
GST#; TIMESTAMP; ABORT/REDO --if REDO--
GST#; TIMESTAMP; <{CST1, r/w}, {CST2, r/w}...>
```

```
GST#; TIMESTAMP; ERROR CODE; TEXT OF REAL PROBLEM
GST#; TIMESTAMP; EXCEPTION CODE; ABORT GST#
GST#; TIMESTAMP; EXCEPTION CODE; AGENT INIT
GST#; TIMESTAMP; EXCEPTION CODE; AGENT TEST OK
GST#; TIMESTAMP; EXCEPTION CODE; AGENT INIT COMP.
```

***FIGURE 4-4***

## 4.4 A typical logging scenario.

The typical scenario for logging follows an Interaction from invocation by a user to the final write to the master log.

The flow of a Global SubTransaction (GST) is from the Interaction Manager to the target local database agent. The GST traverses through an agent step library that will translate each step of the GST into local database transaction instructions.

The Agent creates a Transaction Record Number (TRN) for the GST. The Agent writes the GST information into the ST file at the BEGIN instruction of the GST.

The information in the subtransaction file includes all the local database instructions and the compensating local database transaction instructions.

The following is a typical flow of a Global Transaction and its subtransactions. (NOTE: When something is sent to the ILD from the IM, it is assumed to be logged at that time.)

GT BEGINS: A directory is made for the global transaction

GT EXECUTES: A record is written to the file system with contents similar to: {[GST ID/ HOST ID], [GTID, IAID], Dependency information, R/W information}

GT BEGINS COMMIT: The file system collects the commit information; it will be logged at commit time in the master log.

GST VOTE PHASE:

(1) IM sends to ILD: "RECORD: INITIATING VOTING PHASE"
(2) IM sends to AGENTS: "READY TO COMMIT? ".
(3) AGENTS: YES/NO
(4) IM sends to ILD: "RECORD: (if all AGENTS sent YES to IM) COMMIT DECIDED.
(5) IM ACK (YES/NO) from AGENTS.
(6) IM sends to AGENTS: "COMMIT".
(7) AGENTS ACK COMMIT.
(8) The log records are moved from the file system to the master log. The ILD makes one entry for the entire operation just completed.

At commit time a specification for a complete compensating transaction is written, via the ALD, to the Agent log. This information also includes projected R/W information for the compensating transaction.

The Agent sends the TRN, also known as the compensation log record number, back with the execute message to the Interaction Manager.

After the commit of the Global Transaction the information of the GT will be stored in the respective logs and look similar to the following:

ALD: <TRN:CST1:CST2:...COMTRN>

ILD: <IATRN:{GST1,TRN1:GST2,TRN2:...}COMIATRN>

Special entries of the Logger into the log.

The delimiter/date-time stamp is a special symbol/record that is periodically placed in the log.

The assumption in the above scenario is one in which everything goes according to plan. See the section titled "Handling Events from the Event Manager for logging of events.

### 4.5 Flushing the log.

The log will be periodically flushed in a similar way as a standard garbage collection utility would work. The criterion for flushing the log is that the Interaction has completely committed and all Global Transactions can now be purged from the master log.Summary.

## 5 IRS Scenarios.

### 5.1 A Typical Rollback/Recovery Scenario.

A recovery operation typically consists of two steps. First is the undo of a particular Global Transaction and then the execution of new GSTs. These GSTs could be the same as before, but typically they will differ from the original GST.

The Interaction Manager receives the message that a recovery needs to take place. The IM sends the Transaction ID of the global transaction to be undone to the ILD. It then checks to see if the transaction is active or committed. If it is active it calls a routine within IM to abort the transaction. No further rollback is needed at this point. However, if the transaction is committed the ILD retrieves the necessary log record information using the Global Transaction number and passes it on to the Rollback and Recovery Daemon (RRD) for possible rollback and recovery.

The Interaction Manager recognizes that this "new" GT is actually a recovery operation and sends the information directly to the designated Local Agent. However, during recovery, the IM iterates through the new list of CSTs and sends them out serially.

The Local Agent receives the information (which is basically the log record number in the agent log that contains the Compensating Subtransaction information) and calls the ALD to retrieve the log record. The log record is retrieved and passed back to the Agent which parses it into a set of steps. The step library will use these steps to create the local database instructions.

Once the step library receives the compensating steps the process is treated as any other subtransaction. The step library parses the steps into local database transaction instructions. These instructions are passed to the local database for execution. New compensating steps (for future UNDO operations) are generated and logged as before. This treatment allows the logs to maintain a record of all subtransactions and their undo/redo actions on each of the subtransactions.

## 5.2 Introduction to Termination Scenarios.

The Interactions system has the ability to cause a Global Transaction (GT) or an Interaction (IA) to be cancelled at anytime up to the point the Interaction commits.

An abort in the context of this system is the actual termination of a transaction. Within the action of an abort the system may have to reverse local database transactions. This action is accomplished through semantically undoing that transaction at the local database level.

An abort can occur when the TASL interpreter encounters an abort instruction during the execution of a TASL program. Aborts can also occur when the 2PC protocol fails at some point. Finally aborts can be initiated from some event being raised be the Event Manager (EM).

The action of an abort has one main effect - the dissolution of the specified GT or IA. The termination can be executed on both uncommitted and committed objects. The actions of the Interaction Recovery System (IRS) will be different, dependent only upon the status of the targeted object of termination.

Abort does have side effects, at times, that cause the whole process of termination and cancellation to quickly become an exercise of complication. The key side effect caused by the abort process is the termination, cancellation and possibly undoing of other GTs. This side effect is blatant when the designated object of termination (**DOT**) has a large tree of objects that have been created after the DOT (this will be referred to as the designated object order list or **DOT-L**). As we shall see shortly, this list will consist of uncommitted and committed objects that are either dependent upon the DOT, are a member of the group of objects that are directly descendent from the DOT, and/or are objects that have some specified dependency on a member of the DOT-L.

### Scenarios for Termination.

Let us assume for this discussion that we have an ordered set of GTs that are represented graphically as follows:

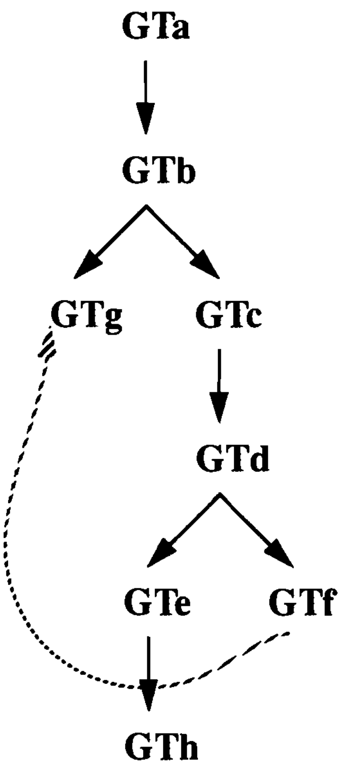


*FIGURE 5-1*

Figure 5-1 illustrates a tree of GTs such that GTa occurs before GTb and GTb occurs before GTc or its dependents, or before GTg. From this illustration we can construct a dependency information structure. This structure consists of the parent GT node and a set of GT nodes that have some dependency ordering information associated to the parent GT node. Note that the set of GT nodes are not necessarily children of the parent. The dependency information for the Figure One is:

**GTa** {}, The set is empty because GTa is the root of the IA.

**GTb {GTa}**, GTb occurs after and is a descendent of GTa.

**GTc {GTb}**

**GTg {GTb, GTf} GTg reads from GTf**

**GTd {GTc}**

**GTe {GTd}**

**GTh {GTe}**

**GTf {GTd, GTg}**

The dependency information is now complete. The information is provided by the TASL interpreter to the IRS via the IM. The IRS creates a dependency file (see Figure 4-1) in the IA(ID#) directory so that it is readily available for termination and cancellation operations. The format of the dependency information then is:

current GT {previous GT, dependent GTn,...}

With the dependency information computed, we can now abort designated GTs and follow the complete action of the IRS during an abort.

Table 1 is a complete log for the Interaction illustrated in Figure 5-1.

**Table 1: Facsimile Log**

| DATE | ID# | Record Type | Text |
|---|---|---|---|
| HHMMDDYY | IA## | IA | BEGIN IA |

## Table 1: Facsimile Log

| DATE | ID# | Record Type | Text |
|---|---|---|---|
| HHMMDDYY | IA##GTa | GTa | BEGIN GT |
| HHMMDDYY | IA##GTa | GT | COMMITTED |
| HHMMDDYY | IA##GTa | GT | GST#,Host, GST#, Host, ... |
| HHMMDDYY | IA##GTb | GT | BEGIN GT |
| HHMMDDYY | IA##GTb | GT | COMMITTED |
| HHMMDDYY | IA##GTb | GT | GST#,Host, GST#, Host, ... |
| HHMMDDYY | IA##GTc | GT | BEGIN GT |
| HHMMDDYY | IA##GTg | GT | BEGIN GT |
| HHMMDDYY | IA##GTc | GT | COMMITTED |
| HHMMDDYY | IA##GTc | GT | GST#,Host, GST#, Host, ... |
| HHMMDDYY | IA##GTd | GT | BEGIN GT |
| HHMMDDYY | IA##GTd | GT | COMMITTED |
| HHMMDDYY | IA##GTd | GT | GST#,Host, GST#, Host, ... |
| HHMMDDYY | IA##GTe | GT | BEGIN GT |
| HHMMDDYY | IA##GTf | GT | BEGIN GT |
| HHMMDDYY | IA##GTe | GT | COMMITTED |
| HHMMDDYY | IA##GTe | GT | GST#,Host, GST#, Host, ... |
| HHMMDDYY | IA##GTf | GT | COMMITTED |
| HHMMDDYY | IA##GTf | GT | GST#,Host, GST#, Host, ... |
| HHMMDDYY | IA##GTg | GT | COMMITTED |
| HHMMDDYY | IA##GTg | GT | GST#,Host, GST#, Host, ... |
| HHMMDDYY | IA##GTh | GT | BEGIN GT |
| HHMMDDYY | IA##GTh | GT | COMMITTED |
| HHMMDDYY | IA##GTh | GT | GST#,Host, GST#, Host, ... |

## Scenario One.

TASL sends the message ABORT(GTh) to the IA. The IA recognizes that this is an abort and passes the message directly to the IRS. The IRS receives the message from the IA, parses it and, at recognizing that it is an abort, enters the termination code.

The termination code reads in the dependency information. It makes a one-pass read through the file and constructs the dependency ordering information (the DOT-L) for GTh (the DOT). In our example the DOT-L would be { } for the DOT and the tree would have only the GTh node on it.

We now call the ILD to search the log for GTh. We are faced with two possibilities. First is that the GT has not committed and therefore is considered an active GT. In this case the IRS returns to the IM instructing the IM to abort GTh. The IM removes the GTh object and returns "success" to the IRS and the IRS then cleans up the IA(ID#) checkpointing directory.

The IRS clean-up of a checkpointing file is accomplished by first removing the GTh from the dependency information file; second deleting the contents of the /IA/GTh directory and deleting the GTh directory itself and, lastly, writing a GTh ABORT record to the ILD log. (The IA must pass to all agents the instruction to delete all of the Global Subtransactions (GSTs) associated with the aborted GT. The IA only returns success to the IRS after it has received success from the agents.) Once the ABORT record has been written to the ILD log, the IRS passes "success" back to the IA and the abort is complete.

The action of the ILD is different when the GT has been committed. When the IRS discovers that GTh is in the ILD log as committed it reads the complete log record of GTh into memory. The Rollback Recovery System (RRS) is then called to prepare the record for an "UNDO". It is at this point where, if we are using a commutation scheme for optimization of the compensating GT we run our commutation routines.

The RRD packages the log record into sequentially ordered GSTs. It passes this package (now a global transaction (GTun)) to the IM. The IM recognizes this as a compensating GT and passes the correct GST (which is actually a log sequence number) to the designated host. (The reader is reminded that the ILD log record consists of GST/HOST pairs.) The agent receives the message from the IA, recognizes it as an UNDO record and immediately passes it to the ALD.

The ALD searches its log for the designated compensating record. This record contains the compensating steps and associated data for the GST. The record has already been reversed so that the compensating steps have been written to the log in the order in which they will be undone. (This is accomplished at the time of the local commit of the transaction. The ALD will log all of the associated compensating steps of a committed transaction at the time of commit.) The ALD packages this record up as a list of steps and returns it to the AGENT. The agent then executes the steps as if they had been received from the IA. From this point on the action of the agent on this GST is like any other GST. The compensating steps are step library calls that generate local database instructions and are executed on the local database. New compensating steps are created and logged as any other set of compensating steps. Once the agent has finished the GST it awaits the 2PC protocol for the GT as any normal GT commit.

The ILD logs two records for this type of cancellation. First, the ILD logs an ABORT record for GTh. This can be considered purely a housekeeping record. In case of serious failures, where human intervention is required, the log can be interpreted by system programmers for debugging purposes. Second, because the UNDO is treated as a new GT, it is logged as any other GT with its member GSTs being logged as a normal operation at the AGENT level.

### Scenario Two.

Please refer to Figure 5-1 for the partial tree of this scenario. The dependency information would look like the following:

**GTe {...}**

**GTh {GTe}**

Now TASL wants to abort GTe. In this scenario, any GT "below" GTe must also be aborted; GTh now is marked for an abort. This can be considered a side effect, as noted previously.

The ILD constructs the dependency information from the information stored in the IA check pointing directory. The one-pass read of the file produces the set (the DOT and DOT-L):

**GTe {GTh}**

For each GT, the ILD now must either get its record from the ILD log (if committed) or determine that the global transactions are not yet committed. (The ILD actually determines the correct place to obtain the information for each individual GT. Also, if GTe is not yet committed, GTh will not be committed and therefor the information would be located in the directory structure) GTh must be terminated first, to guarantee database consistency at the level of occurrence of the GTe termination.

The termination of the GTs (both the DOT and the members of the DOT-L) follows the same operation flow as the termination flow detailed in Scenario One. The deletion of objects, the logging of the aborts, and where the global transactions have been committed, the logging of the new compensating global transactions is consistent with what has already been described.

### Scenario Three.

In Scenario Three TASL requests to ABORT(GTd) (refer to Figure 5-1). The dependency information is as follows:

**GTd {GTc}**

**GTe {GTd}**

**GTh {GTe}**

**GTf {GTd}**

**GTg {GTf, GTd, GTe, GTc}**

The ILD's resulting DOT-L is:

GTd {GTe, GTf, GTg,GTh}

Because there is some state/execution ordering dependency between GTf and GTg, GTg must be included in the DOT-L for the complete list of GTs that will be terminated, cancelled or aborted from the single TASL instruction ABORT(GTd). While at first glance this may seem like overkill for an ABORT, the objective of any rollback scheme is to maintain a consistent set of databases. In order to meet this objective, IRS must ensure that any GT that is affiliated with any other GT within the parent IA be treated as dependent to the that GT. Thus our DOT-L contains not only those GTs that are direct descendents of the DOT but also must contain at GTs that have any associated dependency (typically some state/dependency information internal to the Interaction).

As before, with the DOT-L defined, the IRS follows the same routine as previous scenarios. It calls the ILD which then retrieves the records for the GT that will be terminated. If the GT is not committed, it passes the information back to the IA to destroy the GT. Because the DOT-L contains more then one member, the process iterates through the members until the DOT-L is exhausted, but in inverse order of the DOT-L.

**Scenario Four.**

For this scenario, the IM has received instructions to ABORT an IA. This can occur when exceptions have been raised to the GTs that make running the IA fruitless, the inability of GTs to commit, or effective cancellation of the IA by human intervention. When the IM receives the command **abortIA(IAID)** it is passed to the IRS.

The IRS receives the abort instruction and commences the process of constructing the DOT-L. Because the IA is passed (not a GT) to the IRS, the IRS uses the first GT (in our example GTa) as the DOT. The DOT-L membership then is comprised of all other GTs within the IA. The complete set of dependency members is:

**GTa {GTb, GTc, GTd, GTe, GTf, GTg,GTh}**

If the IA has already been committed it must go to the ILD log for the needed information. A fully committed IA has two records associated with it in the ILD log. First, is the COMMITTED record; second is a copy of the state/execution dependency file for the complete IA.

## 6 Summary.

This paper is a report on the development and design of a logging system for the multidatabase project, Interactions.

While this paper presented the idea of a distributed logging system is necessary in a specific multidatabase, it can be expanded to include other distributed systems and applications. One of the major sticking points on distributed processing is the ability to checkpoint and rollback/recover processes. Applications that use the client/server architecture where one process arbitrates the general operations of the application, but the client application is internally different across all of the clients require specifically tailored checkpointing systems to interface with. While the server may be able to initiate a recovery to a specified point within the local application it would be impossible for the server to know HOW to recover from that given point. It is therefore desirable to have distributed checkpointing systems, as the one described herein, to improve upon checkpointing and subsequent recovering of local applications.

Redo, in the context of Interactions, is not a typical redo. Because of the commutativity properties with the Rollback and Recovery algorithm used in Interactions, redo is considered a re-execution of an interaction from the point of failure or an abort. Because of flexible transactions, re-execution may attempt something quite different from the original transaction. This re-execution is accomplished in the commuted order provided by the Rollback and Recovery algorithm, if commutation is initiated from this particular Global Transaction.

Of special interest is the recovery of some undefined position of the process. This paper only address the concept of a clean break. We either abort a process (global transaction) and start over or we rollback a completed process and start over. It would be of interest to expand the ideas presented here to a more generic method of rollback/recovery which would include restoring a process to a known past state, without loss of integrity.

IRS is a full system that is being implemented by several individuals. The author of this paper was charged with an overall conceptual design and the implementation of the ILD. While it can be reported that the ILD functions, as described herein, there are no present test beds that have the other parts (RRD, IM, Agent, and ALD) to test a full system. This leads to a concern about response time and bottlenecks.

Some improvements or enhancements to the current system could include the following. First the RPC mechanism anticipates synchronisity for all of its client/server interactions. The RPC mechanism could easily be re-worked to utilize the asynchronous paradigm. With this change, both processing time, that is currently time expended on waiting for RPC returns, could be improved. With the umber of RPC waits decreased, more access to the logs would be available.

.The checkpoint files are now on each local system. This requires network traffic and the local system answering requests for access to the checkpoint files. If the checkpoint files were resident on a distributed file server, the speed of access could be improved and therefore less network traffic would probably result.- The local processor would have more time to do database work and the logging functions would be handled by the file server.

# BIBLIOGRAPHY

[BHG87] Bernstein, P.A., Hadzilacos, V., Goodman, N. *Concurrency Control and Recovery in DataBase Systems*. Addison-Wesley Publishing Company. 1987.

[BOO91] Booch, G. *Object Oriented Design with Applications*. The Benjamin/Cummings Publishing Company, Inc. 1991.

[CRI91] Cristian, F., Jahanian. A Timestamp-based Checkpointing Protocol for Long-Lived Distributed Computations. *IEEE Symposium on Reliable Distributed Systems*. 1991.

[ECU88] L'Ecuyer, P., Mallenfant, J. Computing Optimal Checkpointing Strategies for Rollback-Recovery Systems. *IEEE Transactions Computing*. Vol. 37, April, 1988.

[GOL91] Goldberg, A.P., Gopal, A., Lowry, A., Strom, R. Restoring Consistent Global States of Distributed Computations. In *Proceedings of the ACM/ONR Workshop on Parallel and Distributed Debugging*. Pages 144 - 154, May, 1991.

[KOO87] Koo, R., Toueg, S. Checkpointing and Rollback-Recovery for Distributed Systems. *IEEE Transactions on Software Engineering*, Vol. SE-13 (1), January, 1987.

[LI91] Li, Kai, Naughton, J. F., Plank, J.S. Checkpointing Multicomputer Applications. *IEEE Proceedings, Tenth Symposium on Reliable Distributed Systems*. September 30 - October 2, 1991

[LONG91] Long, J., Fuchs, W.K., Abraham, J.A. Implementing Forward Recovery using Checkpointing in Distributed Systems. *Proceedings 2nd IFIP Working Conference on Dependable Computing for Critical Applications*. February, 1991.

[MOH91] Mohan, C. Inderpal, N., Palmer, J. A *case Study of Problems in Migrating to Distributed Computing: Data Base Recovery Using Multiple Logs in Shared Disks Environment*. IBM Research Report. Almaden Research Center, San Jose, CA.

[MOH89] Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwartz, P. *ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging*. IBM Research Report RJ6649, January, 1989. Almaden Research Center, San Jose, CA.

[NOD91] Nodine, Marian H. *Interactions: Multidatabase Support for Planning Actions*. Brown University Technical Report CS-91-64. December, 1991. Computer

Science Department, Brown University, Providence, RI.

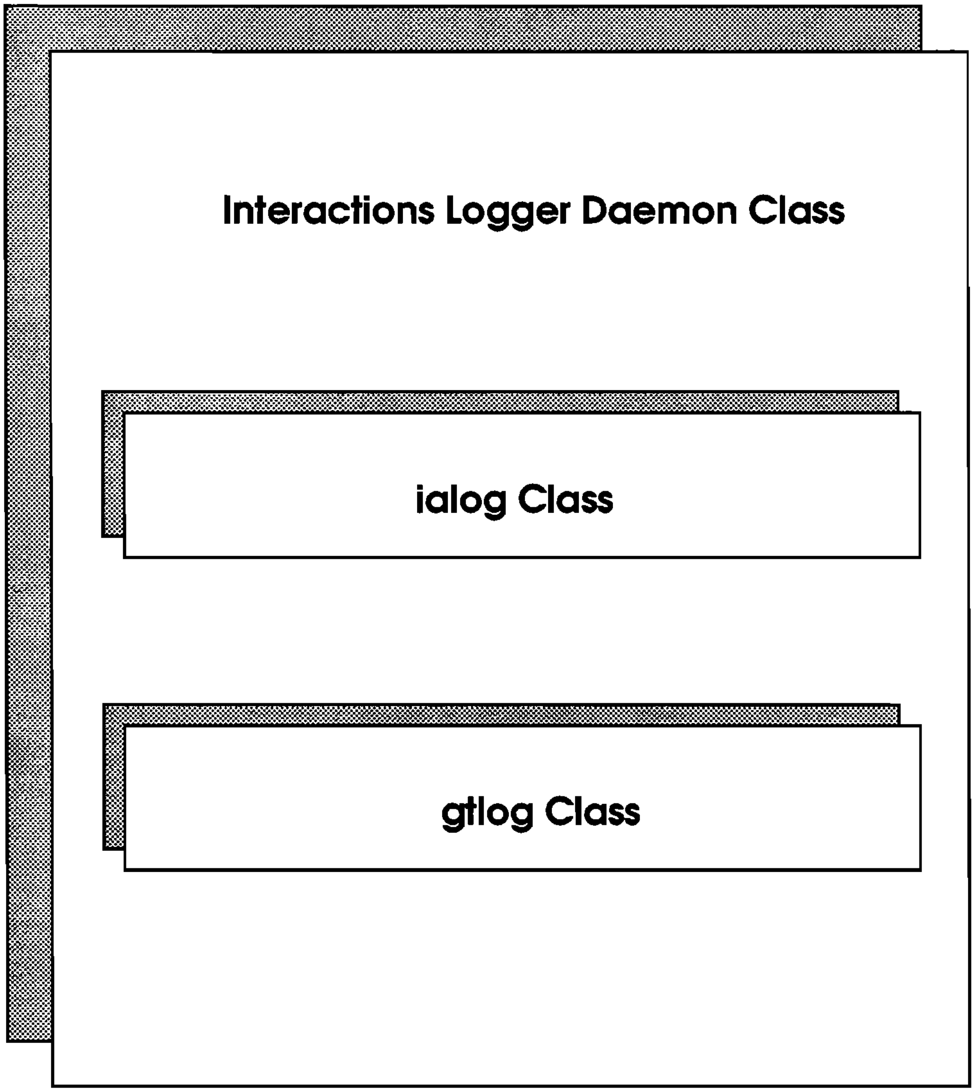
[NOD91-2] Nodine, Marian H. *Supporting Reactive Planning Tasks on an Evolving Multidatabase*. Brown University Technical Report 92-59, December, 1992. Computer Science Department, Brown University, Providence, RI.

# APPENDIX A

# Design Specifications of the Interaction's Recovery System (IRS) for InterActions: A Multidatabase System Approach

Robert T. Baynes,
Department of Computer Science
Brown University, Providence, Rhode Island
7 March 1993

## CLASS DIAGRAM OF IRS



## SUMMARY OF PUBLIC METHODS AND PARAMETERS

| METHOD | PARAMETERS |
|---|---|
| abortIA | IAID |
| retryIA | IAID |
| abortGT | GTID, IAID |
| retryGT | GTID, IAID |
| logIMexception | errno, "text" |
| logIAbegin | IAID |
| logIAcommit | IAID |
| logIAabort | IAID |
| logGTbegin | GTID, IAID |
| logGTcommit | GTID, IAID |
| logGTabort | GTID, IAID |

### Syntax for callIrs with parameters:

callirs(method, $p_1$, $p_2 ... p_n$)

Types for parameters:

IAID - integer
GTID - integer
errno - integer
"text" - string

## SUMMARY OF PRIVATE METHODS AND PARAMETERS

| METHOD | PARAMETERS |
|---|---|
| verifyGTstatus | IAID, GTID |

## SUMMARY OF PROTECTED METHODS AND PARAMETERS

| METHOD | PARAMETERS |
|---|---|
| time_stamp | |
| writeLog | IAID, GTID, type, log_entry |
| checkFile | file name |
| readLog | IAID, GTID |
| delActiveEntry | IAID, GTID |
| createDir | IAID, GTID |
| deleteDir | IAID, GTID |
| GTabort | IAID, GTID, dot-l |
| readSEorderInfo | IAID, GTID |

# Description of Classes

## CLASS ILD

**This class is the parent class of the Interactions Logger Daemon. It provides the interface between the Interactions Manager and the log classes.**

### Data Members:

```
private:

struct SEINFO                    //The structure for seorderInfo
{
          SEDEP se_type;
          int pred;
          int next;
          struct SEINFO *next_se;
          struct SEINFO *prev_se;
} se_info;




struct dotl
{
          int IAID;        // The IAID for this DOTL
          int DOT;        // The start point of the reconstruction
          struct SEINFO *seinfo;
};


struct Log_rec {                 // The log record

    int reccnt;                  //number of records
int record_time[MAXREC];      //time stampe
    int IAID;                 //IAID for this entry
    int GTID;                 //GTID for this entry
    int record_type[MAXREC];  //IA, GT, Exception
    char *log_entry[MAXREC];  //Text of the entry
    int count[MAXREC];        //Length of the entry
 };

  struct Active_rec {         //An active log record
    int record_time;
    int IAID;
    int GTID;
    char *text;
};

public:

int iaerr_sys;                //error number
char * log_name;              //log file name
```

## Member Functions:

```
private:
 int verifyGTstatus(int IAID, int GTID)

protected:

int time_stamp() { return(time(NULL));}
int writeLog(int IAID, int GTID, REC type, char *log_entry)
int checkFile(char* name)
Log_rec *readLog(int IAID, int GTID)
int delActiveEntry(int IAID, int GTID)
int createDir(int IAID, int GTID)
int deleteDir(int IAID, int GTID)
int GTabort( int IAID, int GTID, dotl *rollbck)
dotl *readSEorderInfo(int IAID, int GTID)

public:
ild();
int abortIa(int IAID)
int retryIa(int IAID)
int abortGt(int IAID, int GTID)
int retryGt(int IAID, int GTID)
int logIMException (int IAID, int GTID, REC rectype, Err_Code
err_code)
int logDEpendency (int IAID, int GTID, char *dep_info)
int quickAbort(int IAID, int GTID)
int rollBckGtRrd(dotl *rollb)
int quickAbortIA(int IAID)
~ild()
```

## CLASS iaLog

**This class is a log class of the Interactions Logger Daemon. It provides all methods and data members for the Interaction (IA) logging requirements.**

### Data Members:

```
public:

char iadir_name[255];           //The IA directory Name
int IAID;                       //The interaction ID
int * GTLOG;                    //The ptr to the GT object
iaLog* next_ia;                 // the link to the next IA
```

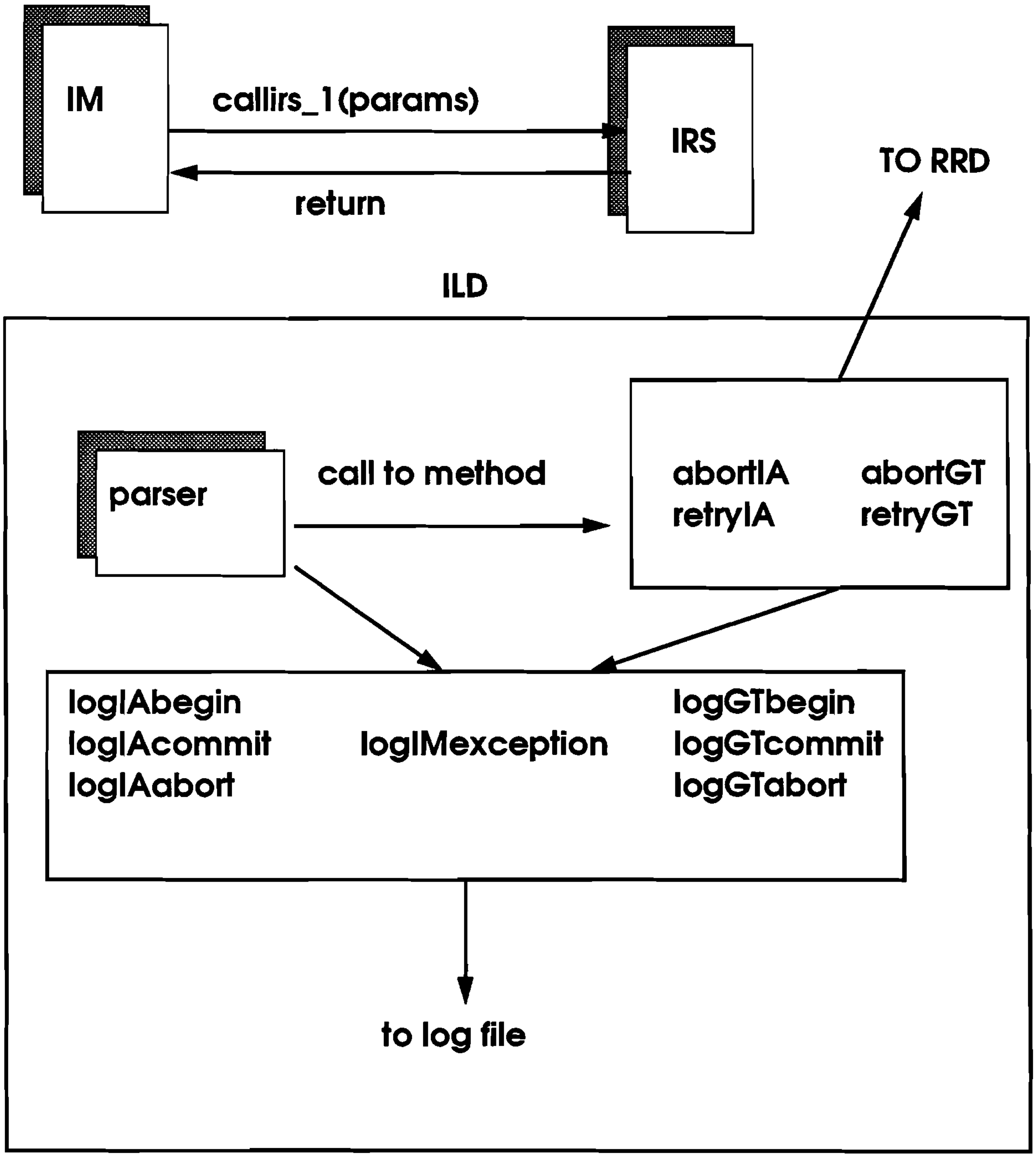
### Member Functions:

```
public:

iaLog ();                       // The constructor for the IA_log

int logIABegin(int IAID);       // logIABegin logs a Begin record to
                                // the ILD log
int logIACommit(int IAID);      // logIACommit logs a Commit record
                                // to the ILD log
int logIAAbort(int IAID);       // logIAAbort logs an Abort  record
                                // to the ILD log

~iaLog ();                      // The destructor for the IA_log
```



Ilustration of Flow of Calls to/From IRS Methods

## CLASS gtLog

### This class is a log class of the Interactions Logger Daemon. It provides all methods and data members for the Global Transaction (GT) logging requirements.

### Data Members:

```
public:

char gtdir_name[255];         //The GT directory Name
int GTID;                     //The global transaction ID
int * GTLOG;                  //The ptr to the GT object
gtLog* next_gt;               // the link to the next IA
int gterr_sys;                //GT error code
```

### Member Functions:

```
public:

gtLog ();                     // The GT constructor

int logGTBegin(int IAID, int GTID, char *depinfo);//logGTBegin
                              // logs a Begin record to the ILD log


int logGTCommit(int IAID, int GTID, char *gst_info);//logGTCommit
                              // logs a Commit record to the ILD
                              // log
int logGTAbort(int IAID, int GTID);     //logGTAbort logs an
                              // Abort record to the ILD log

~gtLog();                     // The GT destructor
```

## INTERFACE DEFINITION FOR Interaction's Recovery System

**INTERFACE:** InterActionManager to Interaction Recovery System

**PURPOSE:** The InterAction Manager (**IM**) provides the key interface to the InterAction's Recovery System (**IRS**). It sends and receives information for logging and recovery.

**LOCATION:** Global Level

**IMPLEMENTATION:**

```
(Parameters: commands to the ILD for
     logging and recovery)
```

These commands conform to the methods defined in the following pages. **The IM does not call or interface directly to the RRD.**

Parameters are passed via RPC calls to the IRS (via callirs_1 call). The command line is passed as a parameter and the ILD method is activated.The IM receives status information from the ILD or Rollback/Recovery record numbers from the RRD. IM must recognize what it is receiving.

**RETURNS:** Status/Error returns from the ILD methods.

## INTERFACE DEFINITION FOR InterAction's Recovery System

**INTERFACE:** ILD to File System

**PURPOSE:** The ILD to File System Interface provides for direct access to the IM log and to the temporary file structure.

**LOCATION:** Global Level

**IMPLEMENTATION:** (Parameters: file/directory descriptors for logging. Once established, data for logging.)

By utilization of standard system calls the file system will be created as any other file system or directory would be created. These calls are used via the I/O streams facilities of C++.

**RETURNS:** Status/error returns from the file status/system routines. Error returns from the methods invoked.

## METHODS DEFINITION FOR Interaction's Recovery System (PUBLIC)

**METHOD NAME:** callirs_1

**PURPOSE:** This method calls the IRS for some procedure to be performed by the IRS

**LOCATION:** IRS

**PARAMETERS:** Procedure to be invoked and its arguments

**INTERFACES:** The IM calls the IRS
The IRS parses the arguments and calls the required method with the given parameters

**RETURNS:** SUCCESS or FAILURE, upon failure, an error code

## METHODS DEFINITION FOR Interaction's Recovery System (c procedure)

**METHOD NAME:** parseIRS

**PURPOSE:** This method parses the parameter line from callirs_1.
It then calls the lld_Cmd routine that will call the
method to be performed by the IRS

**LOCATION:** IRS

**PARAMETERS:** Procedure to be invoked and its arguments

**INTERFACES:** The IM calls the IRS
parseIRS parses the arguments and callslld_Cmds which
then invokes the required
method with the given parameters

**RETURNS:** SUCCESS or FAILURE, upon failure, an error code

## METHODS DEFINITION FOR Interaction's Recovery System (c procedure)

**METHOD NAME:** Ild_Cmds

**PURPOSE:** This procedure calls the method to be performed by the IRS

**LOCATION:** IRS

**PARAMETERS:** Procedure to be invoked and its arguments

**INTERFACES:** The IM calls the IRS
parseIrs parses the arguments and calls ILD_Cmds which in turn invokes the required
method with the given parameters

**RETURNS:** SUCCESS or FAILURE, upon failure, an error code

## METHODS DEFINITION FOR Interaction's Recovery System (PUBLIC)

**METHOD NAME:** `abortIA (IAID)`

**PURPOSE:** This method is called to abort an IA

**LOCATION:** IRS

**PARAMETERS:** `IAID, to be aborted`

**INTERFACES:** The IM calls the IRS
The IRS parses the arguments and starts the abort process

**RETURNS:** SUCCESS or FAILURE, upon failure, an error code

## METHODS DEFINITION FOR Interaction's Recovery System (PUBLIC)

**METHOD NAME:** abortGT (GTID, IAID)

**PURPOSE:** This method is called to abort a GT

**LOCATION:** IRS

**PARAMETERS:** GTID (that is to be aborted)
IAID (of the affected GTID)

**INTERFACES:** The IA calls the IRS using the method callIrs
The IRS parses the arguments and starts the abort process

**RETURNS:** OK if committed; ACTIVE if active

## METHODS DEFINITION FOR Interaction's Recovery System (PUBLIC)

**METHOD NAME:** retryGT (GTID, IAID))

**PURPOSE:** This method is called to retry FROM the passed GT

**LOCATION:** IRS

**PARAMETERS:** GTID
IAID

**INTERFACES:** The IA calls the IRS using the method callIrs
The IRS parses the arguments and starts the retry process using comutation where appropriate. The retry assumes that the GT passed is the DOT from which it will try to redo.

**RETURNS:** SUCCESS/FAILURE, upon failure, an error code

## METHODS DEFINITION FOR Interaction's Recovery System (PRIVATE)

**METHOD NAME:** `verifyGtStatus`

**PURPOSE:** This method verifies a given GT status:
**active**
**committed**

**LOCATION:** IRS

**PARAMETERS:** `GTID`

**INTERFACES:** The IRS calls this method when doing any type of termination of a GT

**RETURNS:** STATUS/FAILURE, upon failure, an error code

## METHODS DEFINITION FOR Interaction's Recovery System (PUBLIC)

**METHOD NAME:** `logIMException ( errno, "text")`

**PURPOSE:** Write Error/Exception records to the IM Log.

**LOCATION:** ILD

**PARAMETERS:** **E type:** `error code and text information`
`exception information`

**INTERFACES:** Called internally within the ILD, information (parameter data) passed from IM to ILD.

**RETURNS:** OK
ERRORS:
- write failure: ACTION: inform IM, IM dies
- write conflict: ACTION: wait

## METHODS DEFINITION FOR Interaction's Recovery System (PUBLIC)

**METHOD NAME:** `logIAbegin (IAID)`<br>`logIAcommit (IAID)`<br>`logIAabort (IAID)`

| | |
|---|---|
| **METHOD NAME:** | `logIAbegin (IAID)`<br>`logIAcommit (IAID)`<br>`logIAabort (IAID)` |
| **PURPOSE:** | Write InterAction records to the IM Log. |
| **LOCATION:** | ILD |
| **PARAMETERS:** | **P1:** `IAID` |
| **INTERFACES:** | Called internally within the ILD, information (parameter data) passed from IM to ILD. |
| **RETURNS:** | OK<br>ERRORS:<br>write failure: ACTION: inform IM, IM dies<br>write conflict: ACTION: wait |

## METHODS DEFINITION FOR InterAction's Recovery System (PUBLIC)

**METHOD NAME:**

```
LogGTbegin (IAID, GTID)
LogGTcommit (IAID, GTID)
LogGTabort (IAID, GTID)
```

**PURPOSE:** Write GT records to the IM Log.

**LOCATION:** ILD

**PARAMETERS:**

```
 P1: GTID
P2: IAID
```

**INTERFACES:** Called internally within the ILD, information (parameter data) passed from IM to ILD.

**RETURNS:**

OK
ERRORS:
write failure: ACTION: inform IM, IM dies
write conflict: ACTION: wait

## METHODS DEFINITION FOR Interaction's Recovery System (PRIVATE)

**METHOD NAME:** `createIADir`

**PURPOSE:** Create interaction temporary directory

**LOCATION:** ILD

**PARAMETERS:** `P1: IAID`

**INTERFACES:** Called internally by ILD
Uses standard system calls for directory creation.

**RETURNS:** OK
ERROR: directory creation failed: ACTION: abort IA

## METHODS DEFINITION FOR Interaction's Recovery System (PRIVATE)

**METHOD NAME:** `createGTDir`

**PURPOSE:** Create global transaction temporary subdirectory

**LOCATION:** ILD

**PARAMETERS:**

```
P1: IAID
P2: GTID
```

**INTERFACES:** Called internally by ILD
Uses standard system calls for directory creation.

**RETURNS:** OK
ERROR: directory creation failed: ACTION: abort IA

## METHODS DEFINITION FOR Interaction's Recovery System (PRIVATE)

**METHOD NAME:** `deleteIADir`

**PURPOSE:** Delete temporary directories when all GT and IAs are committed.

**LOCATION:** ILD

**PARAMETERS:** `Directory file descriptor`

**INTERFACES:** Called internally by ILD, parameters received by IM.

**RETURNS:** OK
ERROR: Directory not empty: ACTION: clean directory before deletion
Directory not found: ACTION: information only.

## METHODS DEFINITION FOR Interaction's Recovery System (PRIVATE)

**METHOD NAME:** `deleteGTDir`

**PURPOSE:** Delete temporary directories when all GTs are committed.

**LOCATION:** ILD

**PARAMETERS:** `Directory file descriptor`

**INTERFACES:** Called internally by ILD, parameters received by IM.

**RETURNS:** OK

ERROR: Directory not empty: ACTION: clean directory before deletion

Directory not found: ACTION: information only.

## METHODS DEFINITION FOR Interaction's Recovery System (PRIVATE)

**METHOD NAME:** `readLog (IAID, GTID)`

**PURPOSE:** Read records from the IM or Agent log, This provides a complete structure of log records for the GT.

**LOCATION:** ILD and ALD

**PARAMETERS:**

```
Log record number:
 IA#
 GT#
 GST#
```

**INTERFACES:** Called by the ILD or ALD after receiving request from IM or Agent.

**RETURNS:** Address in memory of structure containing all records for GTID

ERROR: record not found.

## METHODS DEFINITION FOR Interaction's Recovery System (PRIVATE)

**METHOD NAME:** `readSEorderInfo`

**PURPOSE:** This method reads the state/execution information file and constructs the DOT-L

**LOCATION:** ILD

**PARAMETERS:** `IA; GT to be terminated`

**INTERFACES:** The IRS calls this method when doing any type of termination of a GT. It is called prior to any termination procedure beginning.

**RETURNS:** DOT-L