Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (0223) 323010**
INTERNATIONAL: **+44 223 323010**
FAX: **+44 223 359779**
E-MAIL: **apm@ansa.co.uk**

ANSA Phase III

An Extensible Transaction Framework: Technical Overview

John Warne

Abstract

This report presents a technical overview of the **ANSA extensible transaction framework**: a set of *concepts*, *mechanisms*, and *tools* for constructing and integrating transactional-based business, engineering, and scientific applications in open distributed systems.

The purpose of the framework is to provide a dependable operational environment that supports a wide range of applications which differ widely in their requirements on several aspects of transaction processing and distributed data management, including different transaction models, concurrency control methods, recovery procedures, replication strategies, and timeliness (responsiveness) guarantees.

The framework is also intended to facilitate federated transaction management for heterogeneous, multidatabase systems, in which each distinct system has its own distinct transaction manager.

It is assumed that readers of this report are familiar with the basic architectural principles and concepts of ODP and ANSA.

APM.1060.00.01 **Draft** 25 October 1993

Request for Comments (confidential to ANSA consortium for 2 years)

Distribution:
Supersedes:
Superseded by:

An Extensible Transaction Framework: Technical Overview

Request for Comments (confidential to ANSA consortium for 2 years)

An Extensible Transaction Framework: Technical Overview

John Warne

APM.1060.00.01

25 October 1993

The material in this Report has been developed as part of the ANSA Architecture for Open Distributed Systems. ANSA is a collaborative initiative, managed by Architecture Projects Management Limited on behalf of the companies sponsoring the ANSA Workprogramme.

The ANSA initiative is open to all companies and organisations. Further information on the ANSA Workprogramme, the material in this report, and on other reports can be obtained from the address below.

The authors acknowledge the help and assistance of their colleagues, in sponsoring companies and the ANSA team in Cambridge in the preparation of this report.

Architecture Projects Management Limited

Poseidon House
Castle Park
CAMBRIDGE
CB3 0RD
United Kingdom

TELEPHONE UK	(0223) 323010
INTERNATIONAL	+44 223 323010
FAX	+44 223 359779
E-MAIL	apm@ansa.co.uk

Contents

1	**1**	**Introduction**
1	1.1	Background
2	1.2	Motivation
2	1.3	Structure of the report
3	**2**	**Review of transaction models**
3	2.1	Traditional transaction model
3	2.1.1	Summary
3	2.1.2	Correctness assumptions
3	2.1.3	Benefits
4	2.2	Limitations of traditional transactions
4	2.3	Nested transaction model
4	2.3.1	Summary
5	2.3.2	Benefits
5	2.3.3	Analysis
5	2.4	Saga transaction model
5	2.4.1	Summary
6	2.4.2	Benefits
6	2.4.3	Analysis
7	2.5	Split-join transaction Model
7	2.5.1	Summary
7	2.5.2	Correctness assumptions
7	2.5.3	Benefits
7	2.5.4	Analysis
7	2.6	Cooperative transaction group model
7	2.6.1	Summary
8	2.6.2	Correctness assumptions
8	2.6.3	Benefits
8	2.6.4	Analysis
8	2.7	ACTA model

1 Introduction

This report presents a technical overview of the proposed **ANSA extensible transaction framework**: a set of *concepts*, *mechanisms*, and *tools* for constructing and integrating transactional-based business, engineering, and scientific applications in open distributed systems.

The purpose of the framework is to provide a dependable operational environment which supports a wide range of applications that differ widely in their requirements on several aspects of transaction processing and distributed data management, including different transaction models, concurrency control methods, recovery procedures, replication strategies, and timeliness (responsiveness) guarantees.

The framework is also intended to facilitate federated transaction management for heterogeneous, multidatabase systems, in which each distinct database system has its own private transaction manager.

It is assumed that readers of this report are familiar with the basic architectural principles and concepts of ODP and ANSA.

1.1 Background

The ANSA atomic activity model and infrastructure [WARNE 93], developed and prototyped in ANSA Phase II, specifies an architectural framework for distributed nested transactions based on the traditional properties of atomicity, consistency, independence, and durability (i.e., the so called ACID properties).

The principal objectives of this earlier work were three-fold:

1. to explore the structure and semantics of nested transactions in the context of the ANSA computational model [REES 93];
2. to develop a flexible infrastructure within which to evaluate example mechanisms relating to nested concurrency control, distributed atomic commit protocols, deadlock detection-resolution strategies, and stable version management;
3. to develop and validate a parametric transformer technology for changing the specifications of non-atomic objects (i.e., objects whose operations do not execute atomically in accordance with the ACID properties) to objects whose operations do behave atomically.

This work revealed the power of the nested transaction model in ANSA compliant environments.

The present work extends the ANSA atomicity model and infrastructure to cover a much wider range of transaction processing models and application domains.

1.2 Motivation

Although effective for conventional database applications, the traditional flat transaction[1] with its strict ACID properties is typically found lacking in functionality and performance when used for applications that involve reactive, long-lived, and collaborative activities. Such applications frequently require multiple transactions to share resources in complex ways, for example, by allowing them to exchange access to resources, possibly without any one of them finally committing its execution. Consequently, several extended transaction models have been developed to support the execution of cooperative rather than competitive transactions. In these models, strict adherence to the ACID properties is unacceptable. Instead, the models exhibit properties that allow distinct but associated transactions to share different degrees of visibility, consistency, permanence, and recovery with respect to one another and their use of overlapping resources.

The architectural work described in this report is motivated by the desire to support the requirements of these extended transaction models and their effective application to data management and telecommunications services.

1.3 Structure of the report

To be specified.

1. The degenerate case of the nested transaction.

2 Review of transaction models

Chapter summary to be given here.

2.1 Traditional transaction model

2.1.1 Summary

Traditionally, a transaction is a computation whose execution is constrained by the fundamental properties of atomicity, consistency, independence, and durability (collectively known as the ACID properties [BERNSTEIN 87]). With these properties in place, each transaction is guaranteed to be *failure atomic* and *serializable* with respect to other concurrent transactions. That is, each transaction is guaranteed to execute to completion or not at all; and the effect on system state of the interleaved operations of concurrent transactions is equivalent to the effect of some serial execution of the same transactions. The changes that a transaction makes to system state are made durable if the transaction commits or are discarded if it aborts.

2.1.2 Correctness assumptions

The *traditional transaction model* is based on the assumption that each transaction, when executed alone, always transfers the system from one consistent state to another. The semantics of failure atomicity and serializability ensure similar consistent state transition for concurrent, similarly correct, transactions.

2.1.3 Benefits

There can be no doubt about the overall benefits of the traditional transaction model:

1. its application has proven highly effective for controlling short, concurrent accesses to databases with strict consistency requirements;
2. its failure atomicity and serializability guarantees have shown to significantly simplify the task of programming dependable database applications, especially in distributed environments;
3. its many techniques and algorithms for implementing serializability, failure atomicity, and durability have shown to execute efficiently in commercial database applications.

There can also be no doubt that the use of this model will continue in those application domains where the above benefits are not only vital, but also sufficient. Accordingly, the ANSA extensible transaction framework must provide architectural support for traditional transactions.

2.2 Limitations of traditional transactions

The strong isolation property of the traditional transaction model means that each transaction does not know the identity of other transactions, nor its relative order of execution, since any serialization of concurrent transactions is acceptable. However, either or both of these constraints are not always acceptable for those applications which involve cooperating activities and potentially long-lived computations. Consequently, several extended transaction models have been developed to overcome the limitations of the traditional transaction model. The salient characteristics of a representative set of these extended models are presented below.

Each model is analysed in terms of the following notions:

- **Visibility**, referring to the degree with which one transaction can see the effects of another transaction while it is executing.
- **Consistency**, referring to the correctness of system state that a committed transaction produces.
- **Permanence**, referring to the ability of a transaction to record its results in the permanent state of the system.
- **Recovery**, referring to the ability of transaction, in the event of failure, to recover and take the system to some state that is considered correct.

These properties are referred to as VCPR. It should be noted that the VCPR of a traditional transaction is strictly ACID.

2.3 Nested transaction model

2.3.1 Summary

The *nested transaction model* [MOSS 81] permits transactions to be nested within transactions to form a tree of subtransactions. The root of the tree is called the top-level. This root, which serves as a boundary to encapsulate all descendant subtransactions in the tree, has all the properties of the traditional transaction. Thus the transaction tree as a whole is failure atomic, serializable and, therefore, isolated from other trees.

Each internal subtransaction executes atomically without interference from others and commits or aborts individually. However, if it commits (always as a child), any object resources which it accessed are automatically inherited by its parent for delegation of access to further children or to its parent if it commits. In inheriting resources, a parent does not interfere with its children, only with other conflicting subtransactions which are not descendants of the parent. Each commit action of a child is subsequently made dependent on the commit or abort outcome of its parent. If a parent aborts, its committed children, if any, are also aborted. These commit/abort and resource inheritance strategies are applied recursively and serializably throughout the tree. A final commit decision by the top-level causes all committed descendents in the tree to finally make their effects on all accessed object resources durable.

If a child aborts (voluntarily or involuntarily due to some system failure), the parent is not automatically required to abort. Instead, the parent may be able to initiate its own recovery actions;

1. retry the child subtransaction if the child is potentially recoverable;

2. initiate a contingency child subtransaction to execute an alternative action;
3. abort when any of the above steps cannot be accomplished.

A detailed description of the recovery actions for nested transactions can be found in [WARNE 93].

2.3.2 Benefits

The benefits of the nested transaction model are essentially threefold:

1. **Modularity**: each transaction can be decomposed into a hierarchy of cooperating subtransactions.
2. **Recovery from partial failure**: recovery action can be taken at the granularity of failed subtransactions, rather than necessarily at the root of the tree.
3. **Intra-transaction parallelism**: non-conflicting subtransactions can be executed concurrently.

2.3.3 Analysis

2.3.3.1 Visibility

The visibility characteristics of a nested transaction are expressed in terms of its children. Specifically, each child can view the partial results of its ancestors, the results of its committed siblings, plus any results from other detached committed transactions.

2.3.3.2 Consistency

The top level and its descendant subtransactions are individually failure atomic and serializable and thus a nested transaction as whole operates with the same correctness assumptions as the traditional transaction.

2.3.3.3 Permanence

The ability of a committed child to record its results in the permanent state of the system is dependent on the commitment of its parent. Thus only when the top-level parent of the tree commits and makes its results permanent are all committed descendants allowed to finally make their results permanent.

(Note: a parent is said to hold a *commit dependency* over its children.)

2.3.3.4 Recovery

The unit of failure atomicity for a nested transaction is the subtransaction. Thus when a child fails and is aborted, its parent need not abort, However, if a parent fails and aborts, its descendants are also aborted.

(Note: a parent is said to hold an *abort dependency* over its children.)

2.4 Saga transaction model

2.4.1 Summary

The *saga transaction model* [GARCIA-MOLINA 87] permits a long-lived transaction to be divided into a sequence of subtransactions, each of which has an associated compensating subtransaction that can be triggered to

semantically undo the effects of its committed associate. If a saga subtransaction fails and cannot recover, its partial effects are undone (backward error recovery), and a chain reaction occurs in which any successor committed subtransaction of the same saga are (in reverse execution order) subjected to their respective compensation actions.These compensations do not necessarily return the system state to the point which existed when the saga began.

Thus the composite behaviour of a saga comprising subtransactions S_1, S_2, ..., S_n with respective compensating subtransactions C_1, C_2, ..., C_n is either the sequence

$S_1, S_2, ..., S_n$ (preferable case, where all subtransactions commit)

or the sequence

$S_1, S_2, ..., S_j, C_{j-1}, ..., C_2, C_1$ (for some $2 \leq j \leq n$, where S_j fails and reverts)

The subtransactions of a saga need not observe the same consistent state, since after each commits, another transaction may observe and update overlapping state before the next subtransaction in the saga sequence begins its execution (i.e., saga subtransactions can be interleaved by other transactions, including other sagas). Such interleavings can clearly compromise the serializability of the saga as a whole. However, in practice, at least in a number of telecommunications database applications, such compromises can be endured, and so sagas are applied.

More recently, the saga model has been extended [CHRYSANTHIS 92] to permit:

- sagas that can commit if a non-vital subset of the subtransactions of a saga abort;
- sagas that may contain subtransactions that are not compensatory;
- sagas that may have nested sagas within them;
- sagas that may contain contingency subtransactions for taking alternative actions.

Although these extensions introduce more recovery options, the original principles of the saga model remain essentially the same.

2.4.2 Benefits

Sagas allow a specific way of dealing with the problem of running long-lived transactions, provided compensation can be used as an appropriate recovery technique.

2.4.3 Analysis

2.4.3.1 Visibility

A saga incrementally reveals its partial results to any other transactions, since each successfully executed, intermediate saga subtransaction, commits its result before triggering the next subtransaction in the sequence.

2.4.3.2 Consistency

The correctness assumption for sagas is based on the premise that it is possible to recover system state from committed saga subtransactions by compensation. However, it should be observed that not all transaction executions are compensatory without potential loss of consistency. The

acceptable use of sagas must therefore be evaluated in the light of specific application semantics.

2.4.3.3 Permanence

The ability of each intermediate saga subtransaction to commit and record its result in the permanent state of the system is dependent on the ability of its successor to also commit and record its result without being subjected to subsequent compensation. Thus only when the final subtransaction of the saga commits are the complete set of intermediate results finally made durable.

(Note that the successor of each saga subtransaction is said to hold a *commit dependency* over its predecessor.)

2.4.3.4 Recovery

If any intermediate subtransaction in a saga sequence fails (always resulting in that subtransaction's reversion via backward error recovery), a compensatory sequence is performed that logically removes any previously committed partial results of the saga from the permanent state of the system.

(Note that a successor of each saga subtransaction is said to hold a *compensatory dependency* over its committed predecessor.)

2.5 Split-join transaction Model

2.5.1 Summary

As its name implies, the *split-join transaction model* [PU 88] enables:

1. a transaction to split itself into two independent or dependent transactions;
2. two dependent or independent transactions to join together to form a single transaction.

The specific semantics of these two features are given below.

To be continued.

2.5.2 Correctness assumptions

To be specified.

2.5.3 Benefits

To be specified.

2.5.4 Analysis

To be specified.

2.6 Cooperative transaction group model

2.6.1 Summary

To be specified.

2.6.2 Correctness assumptions

To be specified.

2.6.3 Benefits

To be specified.

2.6.4 Analysis

To be specified.

2.7 ACTA model

To be specified.

References

[BERNSTEIN 87]

Bernstein, P.A., Hadzilacos, V., Goodman, N., “Concurrency Control and Recovery in Database Systems”, Addison-Wesley Publishing Company Inc., 1989.

[CHRYSANTHIS 92]

Chrysanthis, P.K., Ramamritham, K., “ACTA: THe Saga Continues”, Database Transaction Models for Advanced Applications, Editeed by Ahmed K. Elmargarmid, Morgan Kaufmann Publishers, 1992.

[GARCIA-MOLINA 87]

Garcia-Molina, H., Salem, K. “Sagas”, Proceedings of ACM SIGMOD International Conference on Management of Data, 1987.

[MOSS 81]

Moss, J.E.B., “Nested Transactions: An Approach to Reliable Distributed Computing”, MIT/LCS/TR-260, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, U.S.A., 1981.

[PU 88]

Pu, C., Kaiser, G., Hutchinson, N., “Split Transactions for Open-Ended Activities”, IEEE Proceedings of the 14th Conference on VLDB, 1988.

[REES 93]

Rees, R.T.O.R., “ANSA Computational Model”, APM Ltd., Cambridge U.K., April 1993.

[WARNE 93]

Warne, J.P., Rees, R.T.O.R., “ANSA Atomic Activity Model and Infrastructure”, APM Ltd., Cambridge U.K., January 1993.