Volume 4, No. 4, March-April 2013

ISSN No. 0976-5697

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

Constructive Review of Transaction Models in Database Systems

S.Meenakshi*
Associate Professor of Computer Science
Gobi Arts & Science College, Gobichettipalayam
Tamilnadu, India
smgasc@gmail.com

Dr.V.Thiagarasu
Associate Professor of Computer Science
Gobi Arts & Science College, Gobichettipalayam
Tamilnadu, India
profdravt@gmail.com

Abstract: Transaction processing provides mechanisms that can solve many of the problems incurred by concurrent access to shared objects in large databases. The use of transactions to provide reliable and secure information processing and data management has increasingly gained the attention of advanced database applications. Transactions adhering to the ACID properties are guaranteed to be atomic and serializable that is suitable for database applications characterized by short duration transactions and competitive access to shared data. The emergence of advanced applications characterized by long duration with cooperative transactions requires the need for advanced transaction models. Advanced transaction models focus on maintaining data consistency and have provided solutions to many problems such as correctness, consistency and reliability in transaction processing and database management environments. This research paper reviews the various transaction models proposed in the literature that had a considerable impact on the field of transaction processing in database systems. Also this paper identifies that an Event-Condition-Action (ECA) rule paradigm has been a flexible mechanism for supporting the requirements of advanced applications.

Keywords: database; transaction processing; transaction models; ECA rules; database applications

I. INTRODUCTION

Database management systems (DBMS) are at the heart of current information system technology. They provide reliable and effective mechanisms for storing and managing large volumes of information in a multiuser environment. In a multiuser DBMS concurrent execution of transactions is an important aspect of transaction management since it improves the overall system performance by increasing system throughput (the average number of transactions completed in a given time) and response time (the average time taken to complete a transaction). Transactions provide a mechanism for organizing and synchronizing database operations. A transaction is an atomic unit of work that is either completed entirely or not.

A transaction should possess four basic properties called as ACID properties: (i) Atomicity (ii) Consistency (iii) Isolation (iv) Durability and are enforced by the concurrency and recovery mechanisms of the DBMS [32]. In general transactions have been classified based on duration and structure for the effective transaction processing in database applications [30]. Based on duration a transaction may be classified as short duration or long duration transactions. Short duration transactions are characterized by very short execution times in seconds and by competitive access to a relatively small portion of the database. Long duration transactions are characterized by longer execution times in minutes or hours and cooperative access to a larger portion of the database. The division of transactions into subtransactions according to the semantic of applications is called the structure of transactions.

II. CORRECTNESS CRITERIA BASED ON SERIALIZABILITY

Transction processing is concerned with controlling the way in which programs share a common database. When a database is shared and updated by multiple transactions concurrently, serializability is used as a correctness criterion for concurrency control in database applications and it requires that the execution of each transaction cannot be interrupted by other transactions [1, 2, and 3]. Hence to maintain serializability based correctness, the database system through a scheduler has to produce, a schedule of concurrent execution of a set of transactions is equivalent to a serial execution of a set of transactions represented by a serial schedule.

While serializability has been successfully used in applications that are characterized by short duration transactions with competitive access to shared data, it is restrictive and hardly applicable in applications characterized by long duration transactions with cooperative access to shared data since it prevents a transaction from seeing the intermediate results of another transaction [1, 30, and 31]. This limitation has been avoided by the introduction of various nonserializable correctness criteria that extends traditional serializability. The various flexible correctness criteria for concurrency control in databases have been studied and requirements for advanced transaction models are identified [33]. In order to support the requirement of advanced applications, recent researches have proposed the nonserializable correctness criteria with suitable transaction models as a concurrency control mechanism for processing concurrent transactions in databases [18].

Transactions that have strict ACID properties with no internal structure are called flat transaction model and are suitable for handling simple data and applications running with short duration transactions [30, 32]. The flat transaction model is very simple and secure; it lacks the ability to support

applications requiring long living and/or complex and cooperative transactions because of its atomicity and serializability properties since: (i) if a long duration transaction holds a lock on an object, if any other long duration transactions that must access the same object in a conflicting mode must be blocked until the first transaction complete and (ii) if a long transaction cannot complete, all the work that has been done by the transaction must be backed out [16]. The current solution to this problem has been the proposal of extended or advanced transaction models. Hence the focus of the paper is to review the various transaction models that had a considerable impact on the field of transaction processing in database applications.

The rest of the paper is organized as follows. Section 3 briefly reviews the concept of various transaction models proposed in the literature for transaction processing. Section 4 outlines the need for flexible transaction processing due to the requirement of modern applications and section 5 concludes the paper.

III. EXTENDED AND RELAXED TRANSACTION MODELS

A. Nested transaction model:

This model has been designed to extend the flat transaction model to provide the ability to define transactions within other transactions by splitting a transaction into hierarchies of subtransactions [6]. Nested transaction model is a set of subtransactions that may recursively contain other subtransactions forming the complete transaction tree or hierarchy of transactions. The top level transaction can have number of child transactions and each child transactions can also have nested transactions. A child transaction may start after its parent has started, and may commit locally. The committed local result is released only when all of its parents up to the root have successfully terminated. Transactions have to commit from the bottom upwards and a transaction abort at one level does not have to affect a transaction in progress at a higher level. Hence this model is also termed as closed nested transaction. This model is not appropriate for systems that consist of long transactions and it does not address cooperation since full local and global isolation is required. However this model allows increased modularity, finer granularity of failure handling and higher intra transaction concurrency than the flat transaction model.

B. Open nested transaction model:

This model has been proposed to improve the nested transaction model, to relax the isolation requirements by making the results of committed subtransactions visible to other concurrently executing nested transactions [7]. To avoid inconsistent use of the results of committed subtransactions, only those subtransactions that commute with the committed ones are allowed to use their results. Two transactions are said to commute if their effects, their output and final state of the database are the same regardless of the order in which they were executed. This model also relax the condition of commit process occur in a bottom up fashion through the top level transaction as the semantics of these transactions enforce atomicity at the top level. Hence this model permits higher degree of concurrency and cooperation than the nested transaction model and is suitable for systems that consist of long running with cooperative transactions.

C. Multilevel transaction model:

Multilevel transactions are more generalized versions of nested transactions [7]. Subtransactions of a multilevel transaction can commit and release resources before the global transaction successfully completes and commits. If a global transaction aborts its failure atomicity may require that the effects of already committed subtransactions be undone by executing compensating subtransactions. A compensating transaction *T'* semantically undoes effects of a committed subtransaction *T*, so that the state of the database before and after executing a sequence *TT'* is the same. However, an inconsistency may occur if other transaction *S* observes the effects of subtransactions that will be compensated later. Open nested transactions use the commutative to solve this problem.

D. Sagas and Nested Sagas:

Sagas have been proposed as a transaction model for long - lived activities [8, 9]. A saga consists of a set of ACID transactions T_1, T_2,…,Tn with a predefined order of execution, and a set of compensating subtransactions CT_1,CT_2,..,CT_{n-1}, corresponding to T_1,T_2…T_{n-1}. A saga completes successfully, if the subtransactions have committed. If one of the subtransactions, say T_k fails, then committed subtransacitons $T_{1,...}T_{k-1}$ are undone by executing compensating subtransactions $CT_{k-1,}$…CT_1. Each subtransaction is allowed to commit individually. A compensating transaction is then used to explicitly undo its effect if the whole Saga transaction has to abort. By allowing subtransactions to commit, thus revealing their partial result(s), Saga relaxes the full isolation requirement and increase inter-transaction concurrency. Hence some degree of cooperation is permitted. This model has been further extended as a model called Nested Sagas that provide useful mechanisms to structure steps involved within long running transaction into hierarchical transaction structures. This model promotes a relaxed notion of atomicity whereby forward recovery is used in the form of compensating transactions to undo the effects of a failed transaction.

E. ConTract model:

The ConTract model has been proposed to provide a generalized control mechanism for long-lived activities and is aimed at the problem domain of large distributed applications [10]. The basic idea behind this model is to build large applications from short ACID transactions and to provide an application independent system service, which exercises control over them. Also this model does not extend the ACID transactions in structure but embeds them in the application environment and provides reliable execution control over them. In this model a unit of work is defined as a step which ensures the ACID properties but preserves only local consistency. These steps are executed according to a script, which is an explicit control flow description. A reliable and correct execution of the steps is called a ConTract. Hence this model provides control mechanisms like semantic synchronization, context management and compensation at the script level to provide

transaction support to a long-lived and complex application.

F. Split/Join transaction model:

This model has been designed to provide transactions they have ability to share resources by allowing dynamic reconstruction of running transactions [11]. It is suitable for activities with uncertain duration, unpredictable developments, and interaction with other activities. The basic aim of this model is to split a running transaction into two or more transactions and later join other transactions by merging their resources. Also this model allows cooperation among users by allowing transfer of resources from one transaction to other transactions. Further, it uses an adaptive recovery mechanism which allows part of the work done to be recoverable, and since a committing transaction part may release some of its resources, isolation may somehow be reduced. The main drawback of this model is complex merging mechanisms. Also the two resulting transactions from the split command have to obey a serializability criterion which implies that the two transactions must be seen as two isolated transactions while running.

G. Flex transaction model:

This model has been proposed as a transaction model for flexible transaction processing in multidatabase systems [12]. A flex transaction is a set of tasks with a set of functionally equivalent subtransactions for each and a set of execution dependencies on the subtransactions including failure dependencies, success dependencies and external dependencies. Flex transaction model relaxes the atomicity and isolation properties of transactions to provide users increased flexibility in specifying their transactions. To relax the isolation requirement, a flexible transaction uses compensation and relaxes global atomicity requirement allows the transaction designer to specify the acceptable states for termination of the flexible transaction. However scalability and access control are not addressed in flexible transaction.

H. Cooperative transaction hierarchy:

This transaction model has been proposed for design environments and it is a tree based approach similarly like the nested transaction model [13]. The three restricted main levels of this model are: a root, one or more transaction groups and several cooperative transactions. The cooperative transactions correspond to the leaf nodes, which are grouped into transaction groups. They are associated each with a designer in the environment and can, within a transaction group, cooperate on some task. Cooperative transactions are nonserializable and hence for each transaction group, patterns, being a set of rules for how operations can be interleaved, and conflicts, being a set of rules that specify which operations are not allowed to run concurrently, are used as correctness criteria. Although cooperative transaction hierarchy addresses cooperation, its main weakness is the need to define both patterns and conflicts in advance. Hence this model is suitable for applications with a well-defined work structure.

I. CoAct:

Cooperative Activity Model provides the transactional properties applicable to cooperative scenarios. Each user in CoAct works in an own workspace called private workspace and they cooperate through the controlled information exchange and synchronization of their private workspaces [14]. This model works in the following way: a certain parameterized CoAct is used to describe a particular activity and by instantiating it user get a concrete activity. Each participant of a cooperative activity has his/her own activity called user activity and the final result is obtained by merging the result of each user activity. This model is well suited for building asynchronous cooperative applications. But because of the static description of cooperative activity, this model is not flexible enough for advanced cooperative applications.

J. COO:

This model has been developed based on the software development processes requirements with relaxed atomicity and relaxed isolation [15]. Relaxing the atomicity property allows that long transactions may save their intermediate results, thus minimizing losses in the case of crashes and relaxed isolation allows several software processes to access the intermediate results without violating the correctness criterion.

Intermediate results are managed by applying three different object consistency levels such as stable, semi-stable, and unstable. An object is stable when it is fully consistent i.e., a result from a successfully committed transaction. Semi-stable objects are the processes may generate as tentative data, and can be seen as consistent enough, but may violate the correctness criteria. That is, only processes that satisfy the semantic rules and integrity constraints encoded in the software process description are allowed to use semi-stable objects. Unstable objects are those that do not satisfy the correctness criterion at all, and are currently locked by the processes. This object is inaccessible until it becomes stable or semi-stable objects.

K. Evaluation of transaction models:

The transaction models that are reviewed in this paper has been evaluated based on the factors that include transaction properties, transaction structure, intra transaction concurrency, transaction support and the area of application and is shown in Table 1. The presented transaction models are the various extensions to flat transactions that relax the atomicity and isolation properties with the extension of single level structure (flat) to multi level structures. Also the reviewed models have been classified based on the two dimensions such as transaction structure and the structure of objects that they operate on. According to transaction structure the reviewed models use two strategies to achieve different structures inside a transaction: (i) modularize a complex transaction with hierarchies. i.e., a large transaction is divided into smaller components, which can in turn be decomposed and this strategy has been applied in nested transactions, flexible transactions, and open nested transactions (ii) decomposing a long lasting transaction into shorter subtransactions which include a compensation mechanism and this strategy has been applied in sagas, multi level transactions. Along the object structure

dimension the above reviewed transaction models operate on simple objects.

IV. NEED FOR FLEXIBLE TRANSACTION PROCESSING

Database management systems have undergone dramatic changes as a result of the increasing requirements of modern applications. Most of the recent research efforts have addressed newer transaction models due to the transactional requirements of new applications in databases and this rapid growth of transaction models will increase the difficulty of integrating the various models in a uniform manner in a DBMS. Hence today's modern database applications requires a need for: (i) a DBMS to be enhanced with their functionality, to accommodate the requirements of modern database applications ii) a DBMS to be configured to support a flexible transaction management mechanism as needed by the application.

Currently, the choice of the transaction model to be supported by a DBMS is made at the system implementation time there by rendering it difficult to be changed. This is a severe limitation since the support of only one specific transaction model can only serve the requirements of small class of applications. The various existing transaction framework such as ACTA [16], ASSET [17] have been proposed to support multiple transaction models that operate on simple objects with formalized dependencies in conventional database systems.

A. Approach using ECA paradigm:

In order to achieve the above need and the requirement for supporting reactive behavior in database systems, this research paper identifies an effective rule based database paradigm called as active database systems (ADBS) that extend conventional database systems by supporting mechanisms to automatically monitor and react to events that are taking place either inside or outside of the database systems by using active rules [4]. An active rule consists of three components such as an event, a condition, and an action (ECA-rule). The ECA rule paradigm has been efficiently used in advanced application areas such as distributed environments [19, 20], ubiquitous web services [22], event driven computing [23], healthcare system [27], business processes system [29] and e-business applications [24]. The ECA rule paradigm can also give the flexibility to choose transaction models at runtime by activating or deactivating the appropriate rule sets for supporting multiple transaction models. Events and rules can also be reused across rule sets when defining related transaction models [28].

Most of the research and development efforts on active databases and commercial implementations have focused on combining active capabilities in the context of both relational and object oriented database systems [4, 25]. Object oriented database systems provide greater opportunities to model, store and manipulate complex objects with object oriented concepts for complex applications [32]. It has been recognized that many benefits can be gained by integrating active concepts with object oriented database systems [4, 5, 21, and 26].

V. CONCLUSION

This paper has presented a constructive review of transaction models to handle effective transaction processing in database systems. The various transaction models reviewed in this paper operate on simple objects in conventional database systems. The limited support of specific transaction model in conventional database systems and the growing list of today's modern database applications require a need for flexible transaction mechanism to process transactions in advanced databases. The ECA rule paradigm with active capability has been identified as a flexible mechanism for supporting the requirements of advanced applications.

Table: 1 Evaluation of transaction models

Sr. No.	Model	Transaction properties	Intra transaction concurrency	Transaction structure	Transaction support	Application area
1	Flat	ACID	No	No internal structure	Short duration transactions	Applications with competitive access to shared data
2	Nesting	ACID	Yes	Hierarchy of subtransactions	Short duration transactions	Applications with competitive access to shared data
3	Open nesting and Multilevel	A^RCI^RD	Yes	Hierarchy of subtransactions	Short and long duration transactions	Applications with competitive and cooperative access to shared data
4	Sagas and Nested Sagas	ACI^RD	Yes	Sequence of subtransactions with compensating transactions	Long duration transactions	Applications with cooperative access to data
5	Contract	ACID	No	Hierarchy of subtransactions	Long duration and complex transactions	Distributed applications
6	Split/Join	ACI^RD	No	Transactions with dynamic reconstruction	Cooperative transactions	Applications with uncertain duration
7	Flex		Yes	subtransactions with compensating transactions	Cooperative transactions	Multi database applications
8	Cooperative transaction hierarchy	A^RCI^RD	No	Three level tree based approach	Cooperative transactions	Design applications
9	CoAct	ACI^RD	No	Transactions with compensation, dynamic	Cooperative transactions	Asynchronous cooperative applications

				reconstruction		
10	Coo	A^RCI^RD	Yes	Check in and checkout model	Long and cooperative transactions	Software development process

a^r – relaxed atomicity i^r – relaxed isolation

VI. REFERENCES

[1] W.S.Barghouti and G.E.Kaiser, Concurrency Control in Advanced Database Applications, ACM computing surveys, Vol.23, No.3, pp.270-317, 1991.

[2] A.Thomasian, Concurrency Control: Methods, Performance and Analysis, ACM computing surveys, Vol.20, No.1, pp.71-119, 1998.

[3] B. Bhargava, Concurrency Control in Database Systems, IEEE Transactions on Knowledge and Data Engineering, Vol.11, No. 1, pp.3-16, 1999.

[4] N.W. Paton, O. Diaz, Active database systems, ACM Computing Surveys, Vol.31, No.1, pp.63-103, 1999.

[5] J. Campin, N.Paton and M.H.Williams, Specifying Active Database Systems in an Object-Oriented Framework, Software Engineering and Knowledge Engineering, Vol.7,No.1,pp.101-123,1997.

[6] J.E.B. Moss, Nested Transactions: An Approach to Reliable Distributed Computing, Massachusetts Institute of Technology Press, Cambridge, 1985.

[7] G.Weikum and H.J. Schek, Concepts and applications of multilevel transactions and open nested transactions, In A.K.Elmagarmid, editor, Database Transaction Models for Advanced Applications, pp.350-397,Morgan Kaufmann, 1992.

[8] H. Garcia-Molina and K. Salem, Sagas, In Proceedings of the SIGMOD International Conference on Management of Data, pp.249-259, 1987.

[9] H.Garcia-Molina, D. Gawlik, J. Klein, K.Kleissner and K.Salem, Modeling Long-Running Activities as Nested Sagas, IEEE Bulletin of the Technical Committee on Data Engineering, Vol.14, No.1,pp.14-18,1991.

[10] H.Waechter, A.Reuter, The ConTract Model, In A. Elmagarmid (Ed): Database Transaction Models for Advanced Applications, Morgan Kaufmann Publishers, pp. 229-263, 1992.

[11] C. Pu, G.Kaiser and N. Hutchinson, Split transactions for open-ended activities, In Proceedings of the 14th International Conference on Very Large Data Bases, pp.26-37,1988.

[12] A. K. Elmagarmid, Y. Leu, W. Litwin and M. Rusinkiewiczt, A Multidatabase Transaction Model for InterBase, In Proceedings of the 16th International conference on Very Large Data Bases, pp.507-518, 1990.

[13] M.H.Nodine and S.B.Zdonik, Cooperative Transaction Hierarchies: Transaction Support for Design Applications, VLDB Journal, Vol.1, No.1, pp.41-80, 1992.

[14] M. Rusinkiewicz, W. Klas, T. Tesch, J. Wasch and P. Muth, Towards A Cooperative Transaction Model-The Cooperative Activity Model, In Proceedings of the 21st VLDB Conference,pp.194-205,1995.

[15] C. Godart, Coo: A transaction model to support cooperating software developer's coordination, In 4th European Software Engineering Conference, LNCS 717, pp.361-379, 1993.

[16] P. K. Chrysanthis and K. Ramamritham, Synthesis of extended transaction models using ACTA,ACM Transactions on Database Systems, Vol.19, No.3, pp.450-491, 1994.

[17] A.Billris, S.Dar, N.Gehani, H.V. Jagadish, K.Ramamritham, ASSET: A System for Supporting Extended Transactions, In Proceedings of the ACM SIGMOD International conference on Management of Data, pp.44-54, 1994.

[18] A. Buchmann, M.T.Ozsu, M.Hornick, D. Georgakopoulos and F.Manola, A Transaction Model for Active Distributed Object Systems, In A.K.Elmagarmid(ed): Database Transaction Models for Advanced Applications, pp.2-31,1992.

[19] G.Vonbulltzingsloewen, A. Koschel, P.C. Lockemann, H.D. Walter, ECA functionality in a distributed environment, In N.W. Paton, editor, Active Rules in Database Systems, Springer, pp. 147-175, 1999.

[20] S. Chakravarthy, and R. Le, ECA Rule Support for Distributed Heterogeneous Environments, In Proceedings of the 14th International Conference on Data Engineering, IEEE Computer Society,Orlando,Florida,1998.

[21] S.Chakravarthy, V.Krishnaprasad, Z.Tamizuddin, R.H.Badani, ECA Rule Integration into an OODBMS: Architecture and Implementation, Technical Report UF-CIS-TR-94-023, Department of Computer and Information Sciences, University of Florida, Florida, May 1994.

[22] Jae-Yoon Jung, Jonghun Park, Seung-Kyun Han, Kangchan Lee, An ECA-based framework for decentralized coordination of ubiquitous web services, Information and Software Technology,Elsevier, Vol.49, pp.1141-1161, 2007.

[23] K.M. Chandy, Event-driven applications: costs, benefits and design approaches, Gartner Application Integration and Web Services Summit, 2006.

[24] M.Cilia, A.Buchmann, An active functionality service for e-business applications, ACM SIGMOD Record, Vol.31, No.1, pp.24-30, 2002.

[25] E. Simon, A.K.Dittrich, Promises and Realities of Active Database Systems, In Proceedings of the 21st VLDB Conference, pp.642-653, Zurich, Switzerland, 1995.

[26] C.Beeri, T.Milo, A Model for Active Object Oriented Database, In Proceedings of the 17th International Conference on Very Large Data Bases,pp.337-349, Barcelona, 1991.

[27] K. Dube, B. Wu, and J. B. Grimson, Using ECA Rules in Database Systems to Support Clinical Protocols, R. Cicchetti et al. (Eds.): DEXA 2002, pp. 226-235, LNCS 2453, Springer-Verlag, 2002.

[28] E.Anwar, S.Chakravarthy, V.Viveros, Realizing Transaction Models: An Extensible Approach using ECA Rules, Technical Report UF-CIS-TR-95-029, Computer and Information Science and Engineering Department, University of Florida, Florida, 1995.

[29] F.Bry, M. Eckert, P.L. Patranjan, and I.Romanenko, Realizing Business Processes with ECA rules: Benefits, Challenges, Limits, In Proceedings of the International Workshop on Principles and Practice of Semantic Web, pp. 48-62, 2006.

[30] P.A.Bernstein, N.Eric, Principles of transaction processing, second edition, Morgan Kaufmann Publishers, Elsevier, 2009.

[31] R.Elmasri, S.B. Navathe, Fundamentals of Database Systems, 5th edition, Pearson Education Ltd, 2009.

[32] T.M.Connolly, E.Begg, Database Systems: A practical approach to Design, Implementation, and Management, Third Edition, Pearson Education Ltd, 2004.

[33] S.Meenakshi, V.Thiagarasu, Correctness Criteria for Transaction Processing: A Survey and Analysis, International Journal of Applied Research & Studies, Vol. I, Issue. I, Mid-116, ISSN 2278-9480, 2012.