ISSN1000-9825 Journal of Software 软件学报 2000,11(6):720~726

Flexible Cooperative Transaction Model*

MO Qian LI Zi-mu TAN Yu-song ZHOU Xing-ming

(School of Computer Science National University of Defence Technology Changsha 410073)

E-mail: doctor2@nudt.edu.cn

Abstract An advanced transaction model for CSCW applications—FCFM (flexible cooperative transaction model) is presented in this paper. Firstly, the definition of cooperative transactions is given and the states of them are specified. Secondly, the dependencies among cooperative transactions and those between cooperative transactions and outside environments are specified in terms of states of cooperative transactions. Finally, the serializability correctness criterion for the presented model is specified by state dependencies of cooperative transactions. The main advantage of FCTM is that users can customize the states and state dependencies of cooperative transactions to reflect the different requirements of CSCW applications.

Key words CSCW, transaction processing, dependency, serializability correctness criterion.

Traditional database transaction models are usually based on the assumption that numerous independent short period transactions concurrently access a shared database and all the transactions have ACID properties. Recent research has proved that traditional database transaction models and technologies can not support the CSCW applications well. The new requirements in CSCW fields, such as supporting long period transaction, users interaction and cooperation, raise the need for introducing extended transaction models (ETMs).

To overcome the shortcoming of traditional database models and reflect the new requirements of transaction processing for cooperative work environments, extended transaction models have been proposed, such as the Open Nested Transaction Model[1,2], the Sagas Model[3], the CoAct Model[4], etc.

No matter how successful all kinds of ETMs support their original systems, they only reflect some aspects of the relations for a competitive cooperative environment. They are only subsets of relations of complex systems. One ETM proved to be correct in one application may not guarantee its correctness in another application. Some restrictions required by one application may be unacceptable in another application. For example, the Sagas Model can not guarantee database correctness in many banking applications, while the Nested Transaction Model does not allow sharing of uncommitted data crucial in CAD/CAM applications. In a word, an effective extended transaction model in a complex cooperative work environment should be flexible, which means that all kinds of extended transaction processing mechanisms can be applied to reflect different application requirements.

---

* This research is supported by the National Natural Science Foundation of China (国家自然科学基金). **MO Qian** was born in June, 1972. He received his Ph. D. degree in computer science from National University of Defense Technology in 1999. His research interest is distributed database systems. **LI Zi-mu** was born in 1971. He received his Ph. D. degree in computer science from National University of Defense Technology in 1999. His research interest is distributed database systems. **TAN Yu-song** was born in 1975. He is a Ph. D. student at Department of Computer Science, National University of Defense Technology. His research interest is distributed database systems. **ZHOU Xing-ming** was born in 1938. He is a professor and doctoral supervisor of the National University of Defense Technology. His current research areas include super-computer technology and distributed database systems.

Manuscript received 1998-09-15, accepted 1999-01-07.

This paper proposes such a new extended transaction model—the flexible cooperative transaction model (denoted by FCTM). FCTM can not only relax transaction ACID properties but also apply different extended transaction processing mechanisms by customizing states and state dependencies of cooperative transactions to improve performance and guarantee consistency.

The rest of this paper is organized as follows. In Section 1, cooperative transaction concept is defined and cooperative transaction states are specified. In Sections 2, we introduce transaction state dependencies, the key concept of our model. Serializability correctness criterion for our model is described in Section 3. Related work and the advantages of our model are discussed in Section 4.

1 Flexible Cooperative Transaction Concept

To describe complex transaction properties in CSCW applications, we will give the formal description of cooperative transactions and define the states of cooperative transactions.

1.1 Cooperative transaction concept

Definition 1. (Cooperative Transaction) A cooperative transaction is a four-tuple $CT=\{T,O,S,D\}$, where $T=\{T_1,T_2,\ldots,T_n\}$ is a set of sub-transactions or traditional transactions, O is the set of objects CT accesses, S is a set of the states, D is a set of the state dependencies of CT.

There are a lot of long period cooperative transactions in CSCW applications. There exist many complex relations between cooperative transactions, such as multilevel, nested, split, join, etc. Thus, a cooperative transaction can be divided into some sub-transactions and a sub-transaction can have its own constituent sub-transactions, forming a tree structure. The leaves of a cooperative transaction tree are traditional database transactions with ACID properties of local databases. A traditional transaction consists of a serial of read and write operations on the data object set O. Dividing a long period cooperative transaction into a set of sub-transactions and executing sub-transactions concurrently will improve the performance of cooperative transaction processing.

An example that is used throughout this paper is the well-known example from transaction literature: planning a business trip. We consider "planning a business trip from Beijing to New York" as a cooperative transaction CT, which can be a set of such sub-transactions: T_1 reserves a business trip to an agency, T_2 reserves a vehicle from Beijing to New York, T_3 reserves a hotel in New York. Obviously, T_2 and T_3 can be executed concurrently to improve performance. In CSCW field, an application is often composed of many tasks, and a task can be finished with different methods. For example, a task such as traveling from Beijing to New York can be finished with two methods, traveling by a train or by a plane. An efficient cooperative transaction model must allow users to specify the alternative tasks for the same target. For example, T_2 can be divided into two sub-transactions: T_4 reserves a flight from Beijing to New York, T_5 reserves a train from Beijing to New York. T_4 and T_5 are equal at functionality level. The sub-transactions that can finish the same task are called contingency sub-transactions. They form a contingency set of a cooperative transaction.

1.2 Cooperative transaction state

A state set S of a cooperative transaction reflects the cooperative transaction's properties. The definition of cooperative transaction states is as below.

Definition 2. (Cooperative Transaction State) A state set S of a cooperative transaction T at time t can have such values:

$$S=\begin{cases} I & \text{At time } t, T \text{ is in Initial State} \\ E & \text{At time } t, T \text{ is in Executing State} \\ R & \text{At time } t, T \text{ is in Preparing Committing State} \\ C & \text{At time } t, T \text{ is in Committed State} \\ PI & \text{At time } t, T \text{ is in Compensating State} \\ P & \text{At time } t, T \text{ is in Compensated State} \\ AI & \text{At time } t, T \text{ is in Aborting State} \\ A & \text{At time } t, T \text{ is in Aborted State} \\ F & \text{At time } t, T \text{ is in Failure State} \end{cases}$$

I is the initial state of a cooperative transaction, E,R,PI,AI are the intermediate states, and C,P,F,A are the final states (See Fig. 1).

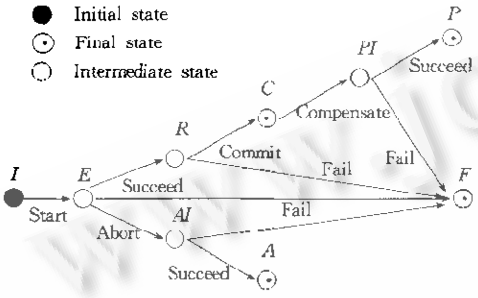

Fig. 1 Cooperative transaction states

In FCTM, the state set S of a cooperative transaction T will be a pre-defined subset of above nine states. Users can customize S according to different requirements of CSCW applications, which is the important flexibility of our model.

For example, traditional two-phase commitment transactions, such as transactions in bank database, can only have these states: $\{I,E,R,C,AI,A,F\}$, but must not have state PI or P.

The atomicity and isolation of traditional transaction model is suitable for short period transactions, which produce a serial of problems that long time resource locking will reduce the concurrent granularity of system and roll-back operation will cause large sacrifice. We introduce compensation transaction concept in FCTM. A cooperative transaction can be compensated, if there exists a sub-transaction which can wipe off the effects of the committed cooperative transaction on a database semantically. If there exists no such sub-transaction, the cooperative transaction is un-compensated. For example, T_4 may have such a compensation transaction: T_6 cancels the flight from Beijing to New York.

2 Flexible Cooperative Transaction State Dependencies

The largest difference between a cooperative transaction and a traditional transaction is that there exist all kinds of relations and interactions among cooperative transactions, between cooperative transactions and outside environment. In this paper we call it dependencies. The cooperation of cooperative transactions is imposed by dependencies. For example, the execution of T_2 is dependent on the commitment of T_1, and the commitment of T3 is dependent on the commitment of T_2. It is just the different dependencies of all kinds of complex transactions that load to a variety of ETMs.

The descriptions of previous ETMs for transaction dependencies are all very simple or pre-defined and do not have the capability to extend to define transaction dependencies.

The FCTM model will specify transaction dependencies in terms of transaction states. Firstly, we give the definition of transaction state dependencies.

Definition 3. (Cooperative Transaction State Dependencies)

The state dependency set D of a cooperative transaction T is defined as a set of 4-element tuple of the form: $\{S_e,T_d,P_r,P_o\}$. S_e is a state in the state set S of the cooperative transaction T, T_d is the cooperative transactions set which S is dependent on, P_r is the previous enable predicate that enforces the cooperative transaction T to be in state S, P_o is the post constraint predicate that the cooperative transaction T must satisfy after it reaches the state S.

Previous enable predicate P_r is all the enable conditions that enforce the cooperative transaction T transit from other state to state S including all kinds of outside and inside events. Post-constraint predicate P_o is used to judge whether the conflict will be caused after the cooperative transaction T reaches state S, whether the consistency of database will be guaranteed.

We will use cooperative transaction state S_e, previous enable predicate P_r, and post constraint predicate P_o to define three simple state dependencies types.

Definition 4. (Backward State Dependencies)

Backward state dependencies between a pair of transactions T_i and T_j impose conditions of the following type: T_i cannot enter state X before T_j has entered state Y. Such dependencies are defined by

$$(X, \{T_j\}, T_i.state = X, Y(T_j) < X(T_i)).$$

$T_i.state = X$ presents the enable condition that enforces T to enter state X. $X(T_i)$ denotes the operation that changes the state of T to X. We use $<$ to specify the order of operations in execution schedule.

Definition 5. (Forward State Dependencies)

Forward state dependencies express conditions of the following type: T_i cannot enter state X after T_j has entered state Y. They are specified by the following:

$$(X, \{T_j\}, T_i.state = X, \neg (Y(T_j) < X(T_i))).$$

Definition 6. (Strong State Dependencies)

Strong state dependencies express conditions of the following type: T_i must enter state X if T_j has entered state Y. They are defined by the following:

$$(X, \{T_j\}, T_j.state = Y, X(T_i)).$$

Combining the three types of cooperative transaction state dependencies with all kinds of states of a cooperative transaction T, we can specify almost all the dependencies defined by recent ETMs. Some common transaction dependencies are specified as follows.

Table 1 Examples of cooperative transaction dependencies

Dependencies	Description	Meaning
Abort	$(A, \{T_j\}, T_j.state = A, A(T_i))$	If T_j aborts then T_i aborts
Execution-on-commit	$(E, \{T_j\}, T_i.state = E, C(T_j) < E(T_i))$	T_i cannot begin execution before T_j commits
Commit-on-termination	$(C, \{T_j\}, T_j.state = C \vee A, C(T_i))$	If T_j terminates then T_i commits
Serial	$(E, \{T_j\}, T_j.state = C \vee A,$ $(C(T_j) \vee A(T_j)) < E(T_i))$	T_i cannot begin execution until T_j either commits or aborts
Begin	$(E, \{T_j\}, T_i.state = E, E(T_i) < E(T_j))$	T_i cannot begin execution until T_j has begun
Force-commit-on-abort	$(C, \{T_j\}, T_j.state = A, C(T_i))$	If T_j aborts then T_i commits
Forward execution-on-commit	$(E, \{T_j\}, T_i.state = E,$ $\neg(C(T_j) < E(T_i)))$	T_i cannot begin execution after T_j commits

Above are all simple state dependencies. We can specify all kinds of complex state dependencies by simple state dependencies. Complex state dependencies consist of simple and possibly other complex state dependencies. For example, a complex dependency "T_4 cannot begin before T_1 commits" and in addition "T_4 cannot begin after T_5 has committed" is specified as follows:

$$\{E, \{T_1, T_5\}, T_4.state = E, [C(T_1) < E(T_4)] \wedge [C(T_5) < E(T_4)]\}.$$

There exist dependencies between cooperative transactions and, additionally, there are dependencies between cooperative transactions and complex outside environments, called outside dependencies. For example, we use time limit t to denote the time deadline for a cooperative transaction T.

Definition 7. (Deadline Dependencies)

Deadline dependencies denote that a cooperative transaction T_i must reach state X before the deadline time t_l, and we specify it by the following:

$$(X, \varphi, T_i.state = X, X(T_i) < t_l).$$

Users can use cooperative transaction state S, previous enable predicate P_e and post constraint predicate P_o to define various kinds of outside dependencies.

3 Serializability Correctness Criterion

A transaction correctness criterion determines the execution schedules that produce correct results and do not violate database consistency. Different ETMs have different correctness criteria. FCTM can specify various correctness criteria by various dependencies to reflect the requirements of real applications. Limited by the space, this paper only specifies the serializability correctness criterion.

3.1 Serializability correctness criterion

Serializability correctness criterion is a conflict-based correctness criterion. Conflict-based correctness criterion considers that operations p_i and q_j performed by two different transactions T_i and T_j conflict if their execution order is "important" for determining whether a transaction schedule H is equivalent to a schedule known to be correct. Traditional serializability correctness criterion considers the two operations p_i and q_j conflict if they are performed on the same object and at least one of them is a write operation. That is, serializability takes into account only syntactic information in determining conflicts (i.e., whether operations are reads or writes). To capture this, in the following we assume the state of-the-object operations have an object-provided property that reveals if the operation changes the state of the object on which it is performed. In particular, we consider that the type of $p_i(o_x)$ is write $(p_i(o_x).op\_type = w)$ if the execution of an operation p_i of transaction T_i changes the state of object o_x without reading it. The type of p is read if p only reads the state of o_x. We also assume that conflicts are defined by a conflict table which is a relation that contains an entry (p, q) for each pair of conflicting operations p and q. Each object o must be associated with a conflict table we refer to as o_x.*conflict_table*. To define that operations p and q supported by o_x conflict, an entry (p, q) should be inserted in o_x.*conflict_table*. If (p, q) is in the conflict table of o_x, p and q conflict. The conflict constraint for the serializability correctness criterion is

$$(p_i(o_x), q_j(o_x)) \in o_x.conflict\_table = \begin{bmatrix} \mathbf{TRUE}\ p_i(o_x).op\_type = w \vee [q_j(o_x).op\_type = w] \\ \mathbf{FALSE}\ others \end{bmatrix}.$$

The specification of serializability correctness criterion for transaction dependencies is based on the following observations:

- conflict defines precedence relations between transactions, and
- serializability is violated if a committed transaction indirectly conflicts with itself, i.e., its precedence relation contains itself.

Let H be a schedule over a set of transactions. H defines an order $<$ on the operation performed by the transactions in τ. Conflicting operations in H define a precedence relation between transactions in τ.

Definition 8. (Direct Serialization Precedence)

A transaction T_i precedes T_j in H with regard to direct serialization dependencies (denoted by $T_i SR_{(\tau,o)} T_j$) iff T issues an operation $p_i(o_x)$ that conflicts with an operation $q_j(o_x)$ of T in H. $p_i(o_x)$ precedes $q_j(o_x)$ in H, and T_i and T_j are non-aborted transactions:

$$T_i SR_{(\tau,O)} T_j = \begin{bmatrix} \mathbf{TRUE}\ \exists\ p_i(o_x) \in T_i, q_j(o_x) \in T_j, T_i, T_j \in \tau, i \neq j, o_x \in O \mid [p_i(o_x) < q_j(o_x)] \\ \wedge [AI \notin S_{T_i}] \wedge [AI \notin S_{T_j}] \wedge [(p_i(o_x), q_j(o_x)) \in o_x.conflict\_table] \\ \mathbf{FALSE}\ others \end{bmatrix}.$$

Definition 9. (Indirect Serialization Precedence)

Indirect Serialization Precedence is defined by the following:

$$T_i SR^n_{(\tau,O)} T_j = \begin{bmatrix} \mathbf{TRUE}\ T_i SR^{n-1}_{(\tau,O)} T_j \vee [\exists\ T_k \in \tau \mid T_i SR^{n-1}_{(\tau,O)} T_k \wedge T_k SR^{n-1}_{(\tau,O)} T_j] \\ \mathbf{FALSE}\ others \end{bmatrix}.$$

Definition 10. (Serialization Precedence Transitive Closure)

Serialization Precedence Transitive Closure is defined by the following:

$$T_i SR^*_{(\tau,O)} T_j \Leftrightarrow \exists\ n \in N \mid T_i SR^n_{(\tau,O)} T_j.$$

Theorem Serializability Correctness Criterion for FCTM. Let τ_c be the set of all committed transactions in H. $\tau_c = \{T \mid C(T_i) \in H\}$. H is serializable if each transaction T_i in τ_c does not SR precede itself, i.e., the transitive closure of the T_i with itself does not hold.

$$\forall\ T_i \in \tau_c \mid \neg [T_i SR^*_{(\tau c,O)} T_i]$$

$$\forall\ T_i \in \tau_c \mid \{C, \tau_c - T_i, T_i.state = C, \neg [T_i SR^*_{(\tau c,O)} T_i]\} \in D_{T_i}.$$

4 Related Work and Conclusion

An automata-based specification of state dependencies is discussed in Ref. [5]. This approach cannot be used to specify correctness criteria. ACTA[6] supports ETM specification and implementation-independent reasoning about transaction-execution correctness and structure. The research of Ref. [7] proposes a framework supporting implementation-independent specification of ETMs described in terms of dependencies between transactions, which our model is extended from. The main difference between dependency specifications and ACTA is that dependency specifications provide reliable true and false evaluation results. The main difference between FCTM and Ref. [7] is that the state set of a cooperative transaction can be customized and extended in FCTM.

FCTM is very flexible in a distributed environment consisting of autonomous and heterogeneous systems. The main advantages of our FCTM model are as follows.

- The state set and state dependencies of a cooperative transaction can be tailored and customized to reflect application requirements, and evolve as application requirements evolve.
- Users can develop new ETMs from existing ones by composing their dependencies.

We have defined a flexible cooperative transaction language based on FCTM and now we are implementing a prototype system of a web-based database cooperative middle-ware to support CSCW database applications on web. This prototype will be used to demonstrate the power and flexibility of our model. The mechanisms of many advanced transaction models, such as the Sagas Model, the CoAct Model etc., can all be applied in it. The details about the prototype and its results will be given in another paper after we finish the prototype system.

References

1 Gustavo Alonso, Divyakant Agrawal, Amr-El Abbadi *et al*. Advanced transaction models in workflow contexts. In: Proceedings of the 12th IEEE International Conference on Data Engineering. New Orleans, 1996. 574～581

2 Dayal U, Hsu M, Ladin R. A transactional model for long running activities. In: Proceedings of the 17th International Conference on Very Large Databases. Barcelona, 1991. 113～122

3 Garcia-Molina H, Salem K. Sagas. In: Proceedings of ACM SIGMOD Conference on Management of Data. 1987. 249～

259

4 Rusinkiewicz M, Klas W, Tesh T *et al*. Towards a cooperative transaction model: the cooperative activity model. In: Proceedings of the 21st International Conference on Very Large Databases. 1995. 194~205

5 Attie P, Singh M, Sheth A *et al*. Specifying and enforcing intertask dependencies. In: Proceedings of the 19th International Conference on Very Large Databases. Dublin, 1993. 131~145

6 Chrysanthis P, Ramamritham K. ACTA: a framework for specifying and reasoning about transaction structure and behavior. In: Proceedings of the ACM SIGMOD Conference on Management of Data. 1990. 211~245

7 Georgakopoulos D, Hornick F. A framework for enforceable specification of extended transaction models. International Journal of Intelligent and Cooperative Information Systems, 1994,3(3):225~253

柔性协同事务模型

莫倩 李子木 谭郁松 周兴铭

(国防科学技术大学计算机学院 长沙 410073)

摘要 提出了一个CSCW领域中的高级事务处理模型—柔性协同事务模型FCTM(flexible cooperative transaction model).首先给出了协同事务的定义,并描述了协同事务的状态,然后从协同事务的状态角度刻画协同事务之间、协同事务与外部环境之间的复杂依赖关系,最后用协同事务的状态依赖描述了可串行化正确性准则. FCTM的优点在于用户能够根据不同的CSCW应用领域的需求,灵活地定义协同事务的状态和状态依赖.

关键词 CSCW,事务处理,依赖关系,可串行化正确性准则.

中图法分类号 TP311