Formal Development of Basic Timestamp Concurrency Control Mechanism using Event-B

Girish Chandra  
Computer Science & Engineering Department  
Institute of Engineering & Technology  
Lucknow, India

Divakar Yadav  
Computer Science & Engineering Department  
Institute of Engineering & Technology  
Lucknow, India

ABSTRACT

Formal methods are mathematical techniques that are used to develop model of complex systems. They provide mathematical proofs for ensuring correctness of model. Through formal methods, it may possible to identify and remove errors at prior stage of development. Event-B is a formal method that is used to develop those systems that can be modeled as discrete transition systems. It rigorously describes the problem and verifies the correctness of model by discharging proof obligations. It performs the consistency checking by preserving invariants of model. In this paper, we have done formal verification of basic time stamp mechanism using Event-B. Basic time stamp is concurrency control mechanism to control concurrent execution of transactions. The main objective of timestamp is to execute transactions such that their execution is equivalent to serial execution in time stamp order.

Keywords  
Formal Methods; Formal Specifications; Event-B; Database; Transaction; Basic Timestamp Mechanism.

1. INTRODUCTION

The tremendous growth of complex system has developed research interest to build system which is free from failures. Due to faulty specification, it may possible that developed system may involve errors that may cause failure of the system. For safety critical system, fault tolerance and reliability are the main features. Therefore, it is highly recommended that system must capture correct specification and it should be rigorously verified during its development. The size of state space is very large for complex system. It is unfeasible to correctly verify every execution path for such system using traditional testing methods.

Formal methods are mathematical techniques that are used for modelling and verification of complex systems [1]. They provide mathematical proofs for ensuring correctness of system. Through formal specifications, the system can be correctly verified at the design stage of the development [2]. Formal specifications are well defined mathematical semantics whose purpose is to model the correct system by formalizing all system requirements while hiding implementation details [3]. Event-B [3] [4] [5] is a formal method that is used to develop those systems that can be modeled as discrete transition systems. Modeling through Event-B supports stepwise development of model. It verifies abstract specification and adds details in refinement steps in order to obtain concrete specifications [4].

In distributed database system, scheduler at each site is responsible for managing concurrent access to data items stored at that site [6]. Concurrency control is activity of controlling concurrent execution of transaction such that data consistency is maintained [7]. Serial execution of transactions provides high consistency to database but it loses the efficiency of system. In serial execution, it may possible that transaction has to wait for longer period of time [6]. Therefore, parallel execution of transactions is preferred. The main difficulty during parallel execution of transaction is how to maintain the consistency. For achieving it, concurrency control synchronizes the concurrent execution of transaction by preventing the modification of data objects when any transaction is accessing it. Synchronization of transaction execution is done either by applying the locks or by using timestamp techniques [7]. The lock based techniques ensures serializability by applying various modes of locks on required data objects. The timestamp based techniques use timestamp ordering to determine the order between every pair of conflicting transactions [7] [8]. In this technique, when a transaction is submitted in the system a unique timestamp is assigned to it. The timestamp of transactions decide the serializability order. For example, if timestamp of any transaction $T_a$ is less than timestamp of other transaction $T_b$ then the system must ensure that produce scheduled must be equivalent to a serial schedule where transaction $T_a$ appears before $T_b$. The timestamp based techniques can be further categorized [7] as basic timestamp based approach, conservative approach and multi-version concurrency control system. We have done formal modeling of multi-version concurrency control system in [9].

In this paper, we have considered basic timestamp based concurrency control [7] for formal development of our model. In this approach, transaction will perform successfully read and write operation on data items if they are not written by some younger transactions. We have done formal verification of basic timestamp mechanism using Event-B as a formal method. The remainder of this paper is organised as follows: section II briefly outline our modelling approach, section III describes informal description about our model and events, section IV presents formal specifications of basic timestamp mechanism. Finally, section V concludes our paper.

2. EVENT-B

Event-B [2] [3] [4] [5] [10] [11] is an event driven based formal method which is an extension of classical B method. It is used to formalize and develop those systems that can be modeled as discrete transition systems. Event-B model contains two basic construct of two types [12] [13]: contexts and machines. Context represents static part of model. It may contain the declaration of sets, constants and axioms. Depending on the requirement set may be carrier set or enumerated set. Axioms are used to represent properties of sets. On the other hand,
machine represents dynamic part of model. It contains the system variables, invariants, theorems, and events. Variables may be sets, numbers, relations, functions etc. Invariants represent properties of model. During the execution of model value of variable gets change but invariants of machine should not be violated. The invariants must be preserved during every execution state of model. The machine may contain theorems which are additional properties derived from invariants. A machine can see one or more context. A machine can be refined by other machine having its own context. An event clause of machine specifies how the states are changed. When any event trigger it changes the state of machine by changing the values of variables. An event is composed of guards and actions. Guards represent necessary condition for an event to occur. When guards of any event become true then corresponding list of actions will be performed.

Event-B defines proof-obligations [14] [15] that must be proved in order to support that machine has retained its properties in form of invariants. During execution of any event, invariants of machine should not be violated. Precisely, for each event we have to prove that defined properties of machine hold before and after of an event execution. We have used Rodin platform [16] [17] to develop our Event-B model. For ensuring correctness of our model, it provides the environment through which all generated proof obligations can be discharged either automatically or interactively.

3. INFORMAL DESCRIPTION ABOUT THE EVENTS
We have considered basic timestamp mechanism [7] for formal development of our model. In this mechanism, when a transaction is submitted, a unique timestamp is assigned to it. The main objective of timestamp is to execute transactions according to their timestamps so that their execution is equivalent to serial schedule. For each data item, largest read and write timestamp is also maintained. A transaction successfully performs read or write operation on a data item if that data item is written by an older transaction. The transaction whose timestamp is lesser than other is known as older transaction. The informal descriptions about the events of our model are as follows:

3.1 Submission of Transaction
Any fresh transaction may submit in the system. When a transaction is submitted a monotonically increasing number is assigned to it. Data items required by transactions are also assigned to transaction. Initially, status of transaction is pending. It will not perform any operation until all required data items become available to it.

3.2 Verifying Conflict With Other Transactions
After submission of transaction, it may possible that required data items are already used by some older transactions. Therefore, transaction will perform conflict checking with other transactions and if conflict is there then transaction will wait for the completion of older transaction. When there is no conflict i.e.: required data items are available then transaction goes into active state.

3.3 Read Operation
The transaction whose status is active can perform read operation on required data items. Read operation is successful if write timestamp of data item is less than transaction timestamp. After successfully reading, timestamp of data item is set to as maximum of current read timestamp and transaction timestamp.

3.4 Write Operation
The transaction may perform write operation on required data item. Writing on any data item is possible if read and write timestamp of that data item is less than transaction’s timestamp. After successfully writing, write timestamp of that transaction is set to as transaction’s timestamp.

3.5 Abort Operation
Read operation on a data item will be aborted if that data item is already been written by some younger transaction. Similarly, a write operation will be aborted if data item is read or written by some younger transaction i.e. transaction which has already perform read or write operation has larger timestamp than transaction which want to do read or write.

4. EVENT-B MODEL OF BASIC TIME STAMP MECHANISM
We start with concurrency control model of basic time stamp mechanism where in the context part DATAITEM, DATAVALE and TRANSACTION are declared as carrier set. The set TRANSTATUS is declared as enumerated set having the values PENDING, ACTIVE, READCOMMIT, WRITECOMMIT and ABORT. The machine part contains variables, invariants and events. The variables, invariants and initialization event are given in Fig.1.

Variables:

- transaction, data, transactionsp, counter, readsp, transdataitem, writetsp, transtatus, activetrans

Invariants:

- inv1: transaction ( TRANSACTION
- inv2: transactionsp : transaction → Natural
- inv3: activetrans ( transaction
- inv4: data : database
- inv5: counter : Natural
- inv6: transdataitem : transaction → Pow(1 (DATAITEM)
- inv7: transtatus : transaction → TRANSTATUS
- inv8: writetsp : DATAITEM → Natural
- inv9: readsp : DATAITEM → Natural

Initialisation

- act1: transaction = ∅
- act2: data = ∅
- act3: transactionsp = ∅
- act4: counter = 1
- act5: transdataitem = ∅
- act6: writetsp = DATAITEM × ∅
- act7: readsp = DATAITEM × ∅
- act8: transtatus = ∅
- act9: activetrans = ∅

End

![Fig. 1. Variables, Invariants and Initialization of Machine](image-url)

The variable transaction represents set of started transactions. The variable data is declared as data ∈ database, where in the context database is declared as:

database ∈ DATAITEM × DATAVALE
submit a unique timestamp is allotted to it (act2). The action act4 specifies that dataitem are assigned to transaction tr. The status of transaction tr is set to as PENDING through act5.

4.3 Conflict_ Checking Event

Data items required by the submitted transaction may be locked by other transactions. This event formalizes conflict checking process (Fig. 3). The guard grd2 specifies that transaction tr is not active transaction. The guard grd3 ensures that status of transaction tr is PENDING. The guard grd4 is written as:

∀t:{tt∈activetrans}\( \land \) transdataitem(tt) =\( \emptyset \)

It ensures that data items required by transaction tr are not conflict with data items used by other active transactions tr i.e. (transdataitem(tr) \( \land \) transdataitem(tt)) will be equal to \( \emptyset \). This event sets the status of transaction tr as ACTIVE.

4.4 Read_ Operation Event

Read_Operation event is given in Fig. 4. The transaction will perform successfully read operation on data item if write timestamp of data item will be less than transaction timestamp.

```
Read-Operation

Any tr, di, sel, readvalue Where
grd1: tr : activetrans
grd2: di : DATAITEM
grd3: di ∈ transdataitem (tr)
grd4: transstatus(tr) = ACTIVE
grd5: writetsp(di) < transactiontsp (tr)
grd6: sel = ((readtsp(di), transactiontsp(tr)))
grd7: sel ≠ ∅
grd8: finite (sel)
grd9: readvalue = dvalue(di)

Then
act1: readtsp(di) = max (sel)
act2: transstatus(tr) = READCOMMIT
```

Fig. 4. Read_Operation Event

The guard grd3 represents that transaction tr wants to read data item di. The status of transaction tr is ACTIVE is ensured by guard grd4. The write timestamp of data item di is less than transaction’s timestamp tr is ensured through grd5. The guard grd6 is written as:

\(sel = ((\text{readtsp} (di), \text{transactiontsp} (tr)))\)

It represents a set sel which contains read timestamp of data item di and timestamp of transaction tr. The guard grd9 returns the value of data item di. The occurrence of this event updates the read timestamp of data item di. The new read timestamp of data item di will be maximum value of current read timestamp of data item di and timestamp of transaction tr (act1). The action act2 sets the status of transaction tr as READCOMMIT.

4.5 Write_ Operation Event

This event formalizes update operation (see Fig. 5). Any active transaction may perform successfully write operation over data item if transaction’s timestamp will be greater than
read and write time stamp of data item. The guard grd7 and grd8 ensure that write and read timestamp of data item di are less than transaction’s timestamp tr, respectively. This event updates data value di (act1). It also updates write timestamp of data item di as transaction timestamp tr (act2) and set the status of transaction tr as WRITECOMMIT (act3).

**Write_Operation ⊨**

Any tr, di, newvv Where
grd1: tr : transaction
grd2: tr : activetrans
grd3: newvv : DATAValue
grd4: di : DATAITEM
grd5: di : transdataitem(tr)
grd6: transstatus(tr) = ACTIVE
grd7: writetsp(di) < transactiontsp(tr)
grd8: readtsp(di) < transactiontsp(tr)

Then
act1: dvalue(di) := newvv
act2: writetsp(di) := transactiontsp(tr)
act3: transstatus(tr) := WRITECOMMIT

Fig. 5. Write_Operation Event

**Write_Abort Event ⊨**

Any tr, di Where
grd1: tr : transaction
grd2: tr : activetrans
grd3: di : DATAITEM
grd4: di : transdataitem(tr)
grd5: transstatus(tr) = ACTIVE
grd6: writetsp(di) > transactiontsp(tr)

Then
act1: transstatus(tr) := ABORT
act2: activetrans := activetrans \ {tr}

Fig. 6. Write_Abort Event

**Read_Write_Abort Event ⊨**

Any tr, di Where
grd1: tr : transaction
grd2: tr : activetrans
grd3: di : DATAITEM
grd4: di : transdataitem(tr)
grd5: transstatus(tr) = ACTIVE
grd6: readtsp(di) > transactiontsp(tr)

Then
act1: transstatus(tr) := ABORT
act2: activetrans := activetrans \ {tr}

Fig. 7. Read_Abort Event

### 4.6 Read_Write_Abort Event

The abortion of any read and write operation will occur if transaction does not succeed either in reading or writing on data item. It will be possible if write timestamp of data item will greater than transaction’s timestamp. The guard grd6 of Fig. 6. Indicates that write timestamp of data item di is greater than transaction’s timestamp tr. Therefore, transaction any of its kind either update transaction or read only transaction will be aborted.

### 4.7 Write_Abort Event

The event Write_Abort is given in Fig. 7. The write operation will also be aborted if required data items are already been read by some younger transaction. The guard grd6 indicates that data item di is already read by younger transaction because read timestamp of data item di is greater than transaction’s timestamp tr (grd6). The action act1 changes the status of transaction tr as ABORT.

### 5. CONCLUSION

The concurrency control can be provided either through locks or timestamps. The timestamp based approaches are categorized as basic timestamp mechanism, conservative approach and multiversion approach. Formal modeling of multiversion concurrency control system using Event-B can be found in [9]. In this paper, we have done formal modeling of basic timestamp mechanism using Event-B. The basic timestamp mechanism ensures that execution of transaction is equivalent to serial execution in timestamp order. In this approach, a transaction will perform read and write operation on any data item if that data item had been last written by an older transaction. For ensuring correctness of system, it is required to verify every execution path of model.

Formal methods are mathematical techniques which provide systematic approach for development and reasoning about complex system. They provide proof based approach to verify the correctness of model. We have considered Event-B formal method for development of our model. We have done verification by preserving invariants. Invariants are constraints on machine variables. During execution of model these invariants should not be violated. Invariant preservation is ensured by discharging proof obligations generated by systems. We have used Rodin platform for formalizing our model. Total 43 proof obligations are generated by system out of which 29 proofs are discharged by the prover of Rodin tool automatically while 14 proofs are discharged interactively. While discharging the proof obligations, it gives clear reasoning about the model. In future, we are planning to formalize conservative approach of timestamp in distributed environment.

### 6. REFERENCES


