Formal Verification of Distributed Transaction Execution in Replicated Database System using Event - B

Pooja Sharma
Pranveer Singh Institute of Technology, Kanpur, U.P. (208020)
U.P.T.U., Lucknow

Raghuraj Singh Suryavanshi
Pranveer Singh Institute of Technology, Kanpur, U.P. (208020)
U.P.T.U., Lucknow

ABSTRACT
Distributed database is a technique in which copy of a database is replicated over the network. A distributed database appears to a client as a single database however in all actuality it is an arrangement of databases distributed on numerous computers or servers. Replication of data in a distributed database system is for enhancing data availability and making data more fault tolerant. Formal methods are used for insight knowledge and refinement of technique to be used and formal methods also helps in understanding how to accomplish those objectives. In our model a coordinator site finds a site with a largest replica number and then broadcast its updated replica to all other sites in a distributed environment.

Keywords
Keywords: Formal Methods, Formal Specification, Event-B, Coordinate site, Participant site, Replication.

1. INTRODUCTION
Distributed database is a kind of virtual database whose portions are physically stored in different places. The clients at any zone can get information at wherever in the system just as the information were all secured at the client's own particular zone. A distributed database management system is the item that arrangements with the distributed databases, and gives a passageway segment that makes this course clear to the client. The objective of a distributed database management system is to control the management of a distributed database in a way that it appears to the client as a centralized database. Distributed transactions are executed in a distributed database environment, where a course of action of related information servers host related information. A distributed transaction contains a course of action of sub transactions, each of which is executed by one information server.

Replication is the process of copying and maintaining database objects in multiple databases that make up a distributed database system. Replication can improve the performance and protect the availability of applications because alternate data access options exist. For example, an application might normally access a local database rather than a remote server to minimize network traffic and achieve maximum performance. Furthermore, the application can continue to function if the local server experiences a failure, but other servers with replicated data remain accessible [1].

According to the model requirement it consists of various events and one of the event is Conflict check in which a site checks that resources needed by requesting site is free or not if it is free then it performs its operation. Another event is Broadcast event in this event coordinator site broadcast a message to every site. All the participant sites receives a message is shown by the participant deliver event and in participant reply event all participant site reply to coordinator with their respective replica numbers. In remote replica update event it updates a site having a maximum value because site with a maximum replica number shows the current value of replica.

The remaining of this paper is summarized as: Section 2 explains about the specification language Event-B and Rodin tool. Section 3 explains about events used in model. Section 4 describes about the model. Section 5 finally concludes the paper.

2. EVENT-B AND RODIN TOOL
The original B method is also known as classic B [2] and its event-based evolution is known as event B or event B method. The event B method [3, 5] reuses the sets and logical notations of the classic B method [4] and creates new notations for providing models based on events. In addition, the refinement of models is a key feature for incrementally developing models from a textually-defined system, while preserving correctness; it implements the proof-based development paradigm. Development of each model requires proofs for invariance and refinement. Operations used in classic B method do not exist in the event B method and these operations in B method are substituted by events in event B. Events modify the system's state (or state variables), by executing an action, only if a guard holds true. During the refinement of classic B method maintenance of operations is compulsory whereas in event-B new events can be introduced for the model refinement. Modification of new variables introduces new proof obligations for ensuring the correctness of refinement. At last we can say that an event B model is a system with a finite number of state variables and a finite number of events. If the system reacts to its environment, the event B model should integrate events of the environment.

Event-B is a method for the stepwise development of programs. The development is mostly top-down and gradually introduces details, rather than starting at the concrete level of writing code. In our model Event-B method is implemented in the Rodin tool. Some other B tools are B tool kit, Atlier B, click n Prove and etc. these tools provide a virtual environment for the generation and discharging the proof obligations.

3. INFORMAL DESCRIPTION OF EVENTS USED IN MODEL
The informal descriptions of events are as follows:

3.1 Start Transaction
Start transaction event keep record of all the transaction that is submitted at any site. The site where transaction is submitted is known as coordinator site for that transaction and it creates an entry of submitted transaction and the objects need by this transaction.

3.2 Conflict Check

It is compulsory to watch that the object required by the submitted transaction is possessed by other dynamic transaction or not. On the off chance that the object is obtained by some other transaction then asking for site needs to hold up until that site (site at present acquiring the object) finishes its execution and discharge that object. Transaction manager is in charge of checking such clashes at the asking for site.

3.3 Commit of Transaction

The transaction can be of two sorts either read only transaction or write or update transaction. On the off chance that if the transaction is read only transaction then it reads the estimation of information object from the coordinator site (where the site is submitted) and if the transaction is update or write transaction then it updates information object and then commits the transaction.

3.4 Broadcast

In distributed environment, the sites in a network communicate with another site by exchanging the messages. Here we are considering full replication in which when a transaction is submitted to a site (coordinator site) then coordinator site broadcast the request message to other sites (Participant sites) in a network. When a participant sites reply to coordinator site with their respective replica numbers.

3.5 Participant_Deliver

In this model of full replication, Event *Participant_Deliver* guarantees that all the request messages from the coordinator site got by participant site. In our model we are not considering time delays between messages send and got and we additionally expect that there is no message disappointment happens.

3.6 Participant_Reply

After all participant sites gets a request message from the coordinator site. Participant sites send an answer message to coordinator site with their particular replica number.

3.7 Coordinator_Delivery

All participant sites send their replica number to the coordinator site and afterward in the wake of getting the replica number coordinator site counts the number of sites from which reply message with a replica number is received.

3.8 Add_Site

The event *Add_Site* adds those new sites in the network that are activated sites or a working sites.

3.9 Remove_Site

The event *Remove_Site* simply removes those sites from the network that are down sites or not working sites.

3.10 IdentifyRecentReplica

This event ensures that atleast "card(SITE)-1"(suppose if there are n sites then atleast from n-1 sites reply message should receive) reply messages should be obtain from the participant sites and then it find out the site with a maximum replica number and then update that site(site with a maximum replica number). Aim behind obtaining the maximum replica number is to ensure the current updated replica.

3.11 Remote_Tran_Submit

After identifying the maximum replica number the transaction is submitted at remote site for performing action requested by the submitted site. This event ensures that the submitted transaction is a subset of *siteactivetransa* and no other transaction is holding the same data object which is required by the submitted transaction. If these constraints are fulfilled then it includes the transaction into the set of *siteactivetransa* and set status of transaction to pending.

3.12 Remote_Replica_update

After the commitment of transaction at local site the update messages are sent to remote sites so that they can also change their replicas.

3.13 CommitWriteTran

The event *commitWriteTran* commits the updated transaction. Updation takes place at the site which has the maximum replica number. After updation it set the status of transaction as commit and removes that transaction from the set of active sites and increment the replica number by one.

3.14 ReadTran

The Event *ReadTran* commits the transaction. Read transaction takes place at the site which has the maximum replica number. After reading of transaction it removes that transaction from the set of active sites.

3.15 AbortWriteTran

Write transaction and read transaction occurs only when the coordinator site receives maximum reply from the participant site and site having maximum replica number and if it does not able to fulfill this criteria then abort transaction event occurs in which it aborts the transaction and removes the site from the set of active site set.

4. FORMAL DESCRIPTION OF EVENTS USED IN MODEL

In a distributed transaction model we assumed that every site is in a distributed environment having the same replicated copy, reliable and provides availability of data. Every site has a replica number associated with it and whenever a site wants to perform read and write operation it submit a transaction to a targeting site. Site where a transaction is submitted is known as a coordinator and other remaining sites in distributed environment are known as participant sites. A site submit a request to the coordinator site and then a coordinator site broadcast a message to all the participant sites with a request message to obtain a replica number from them. All participant sites receive a request message from coordinator site and then they reply with their respective replica number. Coordinator site receives a reply messages associated with a replica number and also counts the number of a sites reply to it. Coordinator site set a criterion for replying sites which ensures that if this much of a participant sites reply to it then it will perform a read or write request and if it not fulfill the criteria then it will abort the transaction. Coordinator site identifies replica number with a maximum value which shows that site have a current replica number and then it sends a message to remote site to perform read or write operation on that site and update a replica number by one if performed action was write otherwise there is no change in replica number.

According to the model requirement it consists of various events. One of the event is Conflict check in which a site checks that resources needed by requesting site is free or not if it is free then it performs its operation. Another event is Broadcast event in this event coordinator site broadcast a message to every site. All the participant sites receives a

message is shown by the participant deliver event and in participant reply event all participant site reply to coordinator with their respective replica numbers. In remote replica update event it updates a site having a maximum value because site with a maximum replica number shows the current value of replica. According to this model the description of variables are given below [6]:

4.1 Trans

Trans variable represents the set of started transactions .

4.2 Sitetransstatus

This variable maps each started transaction at a specific site to *TRANSATATUS*.

4.3 Transeffect

The variable *transeffect* is defined as a total function from *trans* to *update function*.

4.4 Transobject

The variable *transobject* is a total function which maps an transaction to a set of items. The set transobject(t*)* represents the set of data objects read by a transaction *t*. The set of objects written to by *t* will be a subset of *transobject(t)*.

4.5 Replica

Variable replica is modeled as *replica* ∈ *REPLICA* and in the context *REPLICA* is declared as:

$$REPLICA = SITE \rightarrow (OBJECT \rightarrow V\,ALUE)$$

It maps a total function from *SITE* to *value function*. The value function maps a total function from objects to value which means there are no such objects which are undefined in replica located at any site thus all the objects should be defined.

4.6 Siteactivetransa

The variable siteactivetransa represents set of activated transaction at a specific site.

4.7 Siteconflictcheckstatus

The variable *siteconflictcheckstatus* maps each started transaction at any site to *CONFLICTCHECKSTATUS*.

4.8 Coordinator

The variable *coordinator* represents site where the transaction is submitted.

4.9 Sender

The sender is characterized as partial function from *MESSAGE* to *SITE*. The mapping (m ⇀ s) ∈ sender demonstrates that message has sent from the site s.

4.10 Deliver

The variable *deliver* demonstrates that the message is delivered to a dedicated site successfully. A mapping (s ⇀ m) ∈ deliver represents that site s has delivered message m.

4.11 Messagetype

The variable *messagetype* is a total function which assigns type of message to message. *MESSAGETYPE* is an enumerated set which defines type of message, a message can be either reply type or request type.

4.12 Replicano

The variable *replicano* is a total function which maps site to any natural number which is a replica number of that site.

4.13 Messagerepno

The variable *messagerepno* is a partial function from *MESSAGE* to natural number. Variable *messagerepno* returns replica number associated to the site.

4.14 Totalrepsite

The variable *totalrepsite* is a natural number which returns the total number of sites replied to coordinator with their respective replica number.

4.15 Corepinfo

The variable *corepinfo* gives information about the site with their respective replica number.

4.16 Maxrepno

The variable *maxrepno* returns maximum value of replica number.

4.17 Updatedsite

The variable *updatedsite* gives information about the site which is needed to update.

VARIABLES

trans,sitetransstatus,transeffect,transobject,replica,siteactivetransa,siteconflictcheckstatus,

coordinator,sender,deliver,messagetype,replicano,msgrepno,totalrepsite,corepinfo,maxrepno,updatedsite

INVARIANTS

inv1: trans ∈ ℙ(TRANSACTION) *inv2: sitetransstatus ∈ (SITE↔trans)⇸TRANSSTATUS*

inv3: transeffect ∈ trans→((OBJECT⇸VALUE)⇸(OBJECT⇸VALUE))

inv4: transobject ∈ trans→ℙ1(OBJECT) *inv5: replica∈REPLICA* *inv6: siteactivetransa ∈ SITE↔trans*
inv7: siteconflictcheckstatus ∈ (SITE↔trans)⇸CONFLICTCHECKSTATUS

inv8: coordinator ∈ trans→SITE *inv9: sender ∈ MESSAGE⇸SITE*

inv10: deliver ∈ SITE↔MESSAGE *inv11: messagetype ∈ dom(sender)→MESSAGETYPE*
inv12: replicano ∈ SITE→ℕ *inv13: msgrepno ∈ MESSAGE⇸ℕ*

inv14: totalrepsite ∈ ℕ *inv15: corepinfo ∈ SITE↔ℕ* *inv16: maxrepno⊂ℕ*

inv17: updatedsite⊆SITE

EVENTS

INITIALISATION ≙

BEGIN

act1: trans ≔ ∅ *act2: sitetransstatus ≔{ }* *act3: transeffect ≔{ }*
act4: transobject ≔{ } *act5: replica ≔rep0* *act6: siteactivetransa ≔ ∅*
act7: siteconflictcheckstatus ≔ ∅ *act8: coordinator ≔∅* *act9: sender ≔ ∅*
act10: deliver ≔∅ *act11: messagetype ≔ ∅* *act12: replicano≔SITE×{0}*
act13: msgrepno ≔ ∅ *act14: totalrepsite ≔ 0* *act15: corepinfo ≔ ∅*
act16: maxrepno ≔ ∅ *act17: updatedsite ≔ ∅*

END

Fig 1: Variables, Invariants and Initialization of Machine

4.18 Start Transaction Event

Event of a start transaction is given in fig2. Transaction *tt*, site *ss*, updates, objects are the parameters (parameters are the local variables used in event) of event *StartTran* and these parameters are then considered by the guards. Guards state the necessary conditions for an event to occur, all the guards are given in the *WHERE* statement which ensures that all the given statements are true because then only actions are triggered.

New transaction *tt* belongs to set *TRANSACTION* is ensured by guard *grd1* and that transaction *tt* should not belong to started transaction *trans* is ensured by guard grd2. Site *ss* belong to set *SITE* is ensured by guard *grd5*. Site ss at which transaction is submitted becomes coordinator site which is shown by *act6* in fig2.

StartTran ≙

ANY *tt, ss, updates, objects*

WHERE

grd1: tt ∈ TRANSACTION

grd2: updates ∈ ((OBJECT ⇸ VALUE) ⇸

(OBJECT ⇸ VALUE))

grd3: objects ∈ ℙ1 (OBJECT) *grd4: tt ∉ trans*
grd5: ss ∈ SITE

THEN

act1: trans ≔ trans ∪ {tt}

act2: sitetransstatus({ss ↦ tt}) ≔ PENDING

act3: transobject(tt) ≔ objects *act4: transeffect(tt) ≔ updates*

act5: siteconflictcheckstatus({ss ↦ tt}) ≔ pending *act6: coordinator(tt) ≔ ss*

END

Fig2: Start Transactiont of Machine.

Action *act1* shows that transaction *tt* is included into set of started transaction and then we set status of that transaction to *PENDING* as shown in *act2*. We also assign the status of *siteconflictcheckstatus* to pending as shown by *act5*. We assign data object to transaction *tt* and update transaction *tt* is shown by *act3* and *act4* respectively.

4.19 Conflict Check Event

The event *checkconflict* checks the conflicts between transactions as shown in fig3. This event ensures that data object required by the requesting transaction *tt* at the coordinator site *ss* is available at that time or not because there can be a possibility that an object required by a transaction *tt* is acquired by some other active transaction at that time. If the data objects are not used by other transaction then conflict check is completed and we add the transaction of that site to the set of *siteactivetransa.* Guard *grd5*

$$\forall tz \cdot (tz \in trans \wedge (coordinator(tt) \mapsto tz) \in siteactivetransa \Rightarrow transobject(tt) \cap transobject(tz) = \emptyset)$$

depicts that any transaction *tz* and transaction *tt* belongs to the set *siteactivetransa* and the data object required by *tt* is not acquired by the transaction *tt* then check conflict is completed and assign its status complete as shown in *act1* and include transaction to *siteactivetransa* set as done in *act2*.

checkconflict ≙
ANY *tt*
WHERE
grd1: tt ∈ trans
grd2: (coordinator(tt) ↦ tt) ∉ siteactivetransa
grd3: sitetransstatus({coordinator(tt) ↦ tt}) = PENDING
grd4: siteconflictcheckstatus({coordinator(tt) ↦ tt}) = pending
grd5: ∀tz·(tz ∈ trans ∧ (coordinator(tt) ↦ tz) ∈ siteactivetransa ⇒ transobject(tt) ∩ transobject(tz) = ∅)
grd6: {coordinator(tt) ↦ tt} ∈ dom(sitetransstatus)
grd7: {coordinator(tt) ↦ tt} ∈ dom(siteconflictcheckstatus)
THEN
act1: siteconflictcheckstatus({coordinator(tt) ↦ tt}) ≔ complete
act2: siteactivetransa ≔ siteactivetransa ∪ {coordinator(tt) ↦ tt}
END

Fig3: Check Conflict Event of Machine.

4.20 Broadcast Event

The Broadcast Event broadcast the message to all sites (participating site) other than the coordinator site as given in fig4. Transaction *tt*, site *ss*, message *mm* are the parameters of the event. The *grd3* ensures that the site *ss* is coordinator site (i.e is the sending site) .

Broadcast ≙
ANY *ss, mm, tt*
WHERE
grd1: tt ∈ trans
grd2: ss ∈ SITE
grd3: ss=coordinator(tt)
grd4: mm ∈ MESSAGE
grd5: mm∉dom(sender)
grd6: ss↦tt ∈ siteactivetransa
THEN
act1: sender:=sender ∪{mm↦ss}
act2: messagetype(mm) := request
END

Fig4: Broadcast Event of Machine.

Guards *grd5* and *grd6* ensures that message *mm* should not be already send by the sender and transaction *tt* at site *ss* is belongs to set of activated transaction then it performs the action *act1* in which message *mm* is send by site *ss* and assign type of a message to request as done in *act2*.

4.21 Participant Delivery Event

A message broadcast from the coordinator site is received by all participant sites is ensured by

the *Participant_Deliver* Event as shown in fig5. Parameters *tt, ss, mm* of event

Participant_Deliver are then considered by the guards. Guard *grd5* ensures that message *mm* should be send by sender and guard *grd7* ensures that the message *mm* should not yet delivered and guard *grd6* checks if type of a message *mm* is request then action *act1* is performed in which it delivers the message to site *ss*.

Participant_Deliver ≙
ANY *ss, mm, tt*
WHERE
grd1: ss ∈ SITE grd2: tt∈trans
grd3: ss≠coordinator(tt) grd4: mm ∈ MESSAGE
grd5: mm∈dom(sender)
grd6: messagetype(mm)=request
grd7: ss↦mm ∉ deliver
THEN
act1: deliver:=deliver ∪{ss↦mm}
END

Fig5: Participant Deliver Event of Machine.

4.22 Participant Reply Event

After receiving a message *mm* from site *ss* participant site has to reply with a replica number of a site. In event *Participant_Reply grd3* checks that if a type of a message is request and *grd1* checks message *mm* from site *ss* is delivered. If all constraints are true then participant site sends a message with a replica number of a site as shown in *act3* and assign a type of a message to reply as shown in *act2*.

Participant_Reply ≙
ANY *tt, ss, m, mm*
WHERE
grd1: ss↦mm ∈ deliver grd2: m∈MESSAGE
grd3: messagetype(mm)=request
grd4: ss≠coordinator(tt) grd5: m∉dom(sender)
grd6: tt∈trans
grd7: mm∈dom(sender)
THEN
act1: sender:=sender ∪{m↦ss} act2: messagetype(m):=reply
act3: msgrepno(m):=replicano(ss)
END

Fig6: Participant Reply Event of Machine.

4.23 Coordinator Delivery Event

Event coordinator delivery is shown in fig7. *Coordinator_Delivery* event models reception of reply messages from participant sites. If a type of a message is reply (as shown in *grd2*) then it receives a message as shown in *act1*. In *act2* it gives total number of site replied with their replica number. In *act3* as shown in figure it assigns a replica number of a site to a message *m*.

Coordinator_Delivery ≙
ANY *tt, ss, m, s*
WHERE
grd1: m∈MESSAGE grd2: messagetype(m)=reply
grd3: tt∈trans grd4: ss=coordinator(tt) grd5: ss↦m ∉ deliver grd6: s≠coordinator(tt)
grd7: m↦s∈sender
THEN
act1: deliver:=deliver ∪{s↦m} act2: totalrepsite:=totalrepsite+1
act3: corepinfo:=corepinfo ∪{s↦msgrepno(m)}

END

4.24 Identify Recent Replica Event

Event *IdentfyRecentReplica* is shown in fig8. In event *IdentfyRecentReplica* grd3 ensures that atleast "card(SITE-1)" sites reply with their replica numbers for performing the operations requested by the transaction. the guard grd4 ensures that the replica with a maximum number should be from the range of *corepinfo* and grd5 ensures that site with a maximum replica number should be from *corepinfo*. If all the above guards true then action takes place, in act1 a set of maximum value of replica is assigned to a new variable maxrepno and in act2 we update a transaction with a maximum replica number.

IdentifyRecentReplica ≙
ANY *s, maxval*
WHERE
grd1: s∈SITE grd2: maxval∈ ℕ
grd3: totalrepsite=card(SITE)−1
grd4: maxval=max(ran(corepinfo))
grd5: s↦maxval∈corepinfo
THEN
act1: maxrepno≔{maxval}
act1: updatedsite≔updatedsite ∪{s}
END

Fig8: Identify Recent Replica Event of Machine.

4.25 Transaction submission at Remote Site Event

The event *Remote_Tran_Submit* as shown in fig9. models the submission of transaction at remote sites for changing the replica number. The guard *grd4*

$$\forall tx\cdot(tx\in trans\wedge(ss\mapsto tx)\in siteactivetransa \Rightarrow transobject(tt)\cap transobject(tx)=\emptyset)$$

ensures that the data objects which are required by transaction *tt* are available at site *ss*. The transaction is activated at site *ss* through *act1* and status of transaction is set to *PENDING* through *act2*.

Remote_Tran_Submit ≙
ANY *ss, tt*
WHERE
grd1: ss ∈ SITE grd2: tt∈trans
grd3: ss≠coordinator(tt)
grd4: ∀tx·(tx∈trans∧(ss↦tx)∈siteactivetransa ⇒ transobject(tt)∩transobject(tx)=0)
THEN
act1: siteactivetransa≔siteactivetransa ∪ {ss↦tt}
act2: sitetransstatus({ss↦tt})≔PENDING
END

Fig9: Transaction Submission at Remote Site Event of Machine.

4.26 Commit Transaction Event

The event *CommitWriteTran* models the commitment of update transaction as shown fig10. The transaction is in active state and it is updated transaction is ensured by *grd4* and *grd7* respectively. This event updates the replica at coordinator site *act2* and set the status of transaction as *COMMIT act1* and after the commitment of transaction it removes the transaction *tt* from the set of *siteactivetransa act3*. After the successful commitment of the transaction *tt* the replica number is incremented by one as shown in *act4*.

commitWriteTran ≙
ANY *tt, pdb, ss*
WHERE
grd1: ss∈updatedsite grd2: tt ∈trans
grd3: pdb=transobject(tt)◁replica(ss)
grd4: ss↦tt ∈ siteactivetransa
grd5: {ss↦tt}∈dom(sitetransstatus)
grd6: sitetransstatus({ss↦tt})=PENDING
grd7: ran(transeffect(tt))≠{∅}
grd8: pdb∈dom(transeffect(tt))
THEN
act1: sitetransstatus({ss↦tt}) ≔ COMMIT
act2: replica(ss) ≔ replica(ss)+transeffect(tt)(pdb)
act3: siteactivetransa≔siteactivetransa \{ss↦tt}
act4: replicano(ss)≔replicano(ss)+1

END

Fig10: Commit Transaction Event of Machine.

4.27 Abort Transaction Event

AbortWriteTran event is given in fig11. Guard *grd3* ensures that a transaction is in *siteactivetransa* and status of a transaction is *PENDING* as shown in guard *grd5*. Guard *grd7* transaction *tt* is an updated transaction then as an action it aborts the transaction *tt* and removes it from the *siteactivetransa* set as shown in *act1* and *act2*.

4.28 Read Transaction Event

ReadTran event as shown in fig12. Models the commitment of read only transaction where transaction *tt* is read only transaction is ensured by *grd6*. This event performs reading of

AbortWriteTran ≙
ANY *tt, ss*
WHERE
grd1: tt ∈ trans grd2: ss∈updatedsite
grd3: (ss↦tt) ∈ siteactivetransa
grd4: {ss ↦ tt}∈dom(sitetransstatus)
grd5: sitetransstatus({ss↦tt}) = PENDING
grd6: ran(transeffect(tt))≠{∅}
THEN
act1: sitetransstatus({ss↦tt})≔ ABORT
act2: siteactivetransa≔siteactivetransa \{ss↦tt}
END

Fig11: Abort Transaction Event of Machine.

the objects from the replica located at coordinator site. It set the status of transaction as *COMMIT act1* and then removes the transaction from *siteactivetransa* set as shown in *act2*.

ReadTran ≙
ANY *tt, readval, ss*
WHERE
grd1: tt∈trans *grd2: ss∈updatedsite*
grd3: (ss↦tt) ∈ siteactivetransa
grd4: {ss↦tt}∈dom(sitetransstatus)
grd5: sitetransstatus({ss↦tt})=PENDING
grd6: ran(transeffect(tt))={∅}
grd7: readval=transobject(tt)◁replica(ss)
THEN
act1: sitetransstatus({ss↦tt}):=COMMIT
act2: siteactivetransa:=siteactivetransa \{ss↦tt}
END

Fig12: Read Transaction Event of Machine.

4.29 Updating Replica at Remote Transaction

The event *RemoteReplicaUpdate* is given in fig13. This event updates the replica at remote site due to execution of transaction *tt*. Site *ss* is not coordinator site for the transaction *tt* is ensured by *grd3*. Transaction *tt* is activated at site *ss* and its status is *PENDING* is ensured by *grd4* and *grd6* respectively.

In this event *remotereplica* is partial database and defined as:

$$remotereplica = transobject(tt) \triangleleft replica(ss)$$

This event sets the status of transaction as COMMIT as shown in *act1* and then removes the transaction from *siteactivetransa* set as shown in *act2*.

$$siteactivetransa := siteactivetransa \setminus \{ss \mapsto tt\}$$

This event updates the replica at remote site act1, set the status of *tt* at site *ss* as *COMMIT act2* and remove the transaction from *siteacivetransa act3*.

Remote_Replica_update ≙
ANY *s, ss, tt, remotereplica*
WHERE
grd1: ss ∉ updatedsite *grd2: tt∈trans*
grd3: ss↦tt∈ siteactivetransa
grd4: {ss ↦ tt}∈dom(sitetransstatus)
grd5: sitetransstatus({ss↦tt})=PENDING
grd6: remotereplica ∈ dom(transeffect(tt))
grd7: remotereplica=transobject(tt) ◁ replica(ss)
grd8: s ∈ updatedsite
grd9: sitetransstatus({s↦tt})= COMMIT
THEN
act1: sitetransstatus({ss↦tt}):=COMMIT
act2: replica(ss) := replica(s)
act3: siteactivetransa:=siteactivetransa \{ss↦tt}
act4: replicano(ss):=replicano(s)
END

Fig13: Update Transaction at Remote Site Event of Machine.

4.30 Add Site Event

Add_Site event as shown in fig14. This event adds site *ss* to a set of a live sites set if a site is from a set *SITE.*

Add_Site ≙
ANY *ss*
WHERE
grd1: ss ∈ SITE
THEN
act1: *livesites := livesites ∪ {ss}*
END

Fig14: Add Site Event of Machine.

4.31 Remove Site Event

Remove_Site event as shown in fig15 removes site *ss* from a set of live sites if site ss in a set of live set.

Remove_Site ≙
ANY *ss*
WHERE
grd1: ss ∈ livesites
THEN
act1: *livesites:=livesites\{ss}*
END

Fig15: Remove Site Event of Machine.

5. CONCLUSION

Formal method is one of the techniques which help us understanding the complex specification and how to achieve those goals. This model uses formal method for the verification of transaction execution by using formal specification language event-B. Event-B is a formal technique that is used for specifying and reasoning about complex systems. Rodin tool is a platform where verification of the program is done. In our model when an update transaction is submitted to a site doesn't perform update operation until unless it finds out the latest site with a maximum replica number. When a transaction is submitted to a coordinator site it broadcast a message to all participant sites, and then all participant sites reply to coordinator site with their replica number and then coordinator site performs update operation on a site with a maximum replica number. After updation it increments the count of replica number by one and if the operation was read then replica number will remain as it is. In our model total eighty proof obligations are generated by rodin tool, out of which seventy two proof are discharged automatically while eight proof requires interaction with the system. This model gives clear insight about the verification of transaction execution in replicated database system.

6. REFERENCES

[1] http://docs.oracle.com/cd/A59447_01/nt_804ee/doc/database.804/a58227/ch_repli.htm

[2] http://www.loria.fr/~mery/erasmusmaynooth/n1.pdf

[3] Jean-Raymond Abrial and Louis Mussat. Introducing dynamic constraints in B In Bert [BER 98], pages 83–128.

[4] Jean-Raymond Abrial. The B book. Cambrige University Press, 1996.

[5] Jean-Raymond Abrial. B #: Toward a synthesis between Z and B In Bert et al. [BER 03], pages 168–177

[6] Raghuraj Suryavanshi, Divakar Yadav "Rigorous Design of Lazy Replication System Using Event-B"