Formal Development of Fault-Tolerant Majority Based Replica Control Protocol using Event-B

Anupam Singh, Raghuraj Suryavanshi, Divakar Singh Yadav

Abstract*: In distributed environment, data availability and concurrency control both are challenging issues. Data availability can be maintained by replicating data at several locations or sites that will improve the availability but at the same time it is very challenging task to maintain the consistency of it. In order to improve the performance of the system, it is required to execute multiple transactions concurrently on several sites. Therefore, we need to control these concurrent transactions for maintaining consistency of replica. Replica control become more complex for the environment where messages are delayed due to communication failure. In this paper, we develop formal model of fault-tolerant replica control protocol Using Event-B. Formal methods are mathematical techniques through which we can verify the correctness of model. Event-B is a formal method which is used to develop the model in distributed environment.

***Index Terms*: Formal Methods, Formal Verification, Event-B, Replication, Replica control Protocol.**

I. INTRODUCTION

Highlight A Distributed system is a collection of autonomous computers which are placed at different locations and connected among themselves through a network [1][2]. Data management at single site or node is easy whereas it is difficult when data is placed at different locations. The major problems which are associated with data management are data availability and consistency [1][2]. It was observed that Data availability can be improved through replication mechanism. Data management will be more complex when it is replicated at several places. Replica controlling is a big challenge to achieve data consistency in distributed system. Replication strategies [3][4][5][6] can be categorized as: optimistic and pessimistic. Optimistic replication protocol is also known as lazy replication [3][5] in which replica may be inconsistent for some time but at the end it will be verified while Pessimistic replication is more conservative because an update cannot be written if a lock is not available. Data availability will be sacrificed using pessimistic approach [4]. Replication can be also classified as partial and full replication [4][6]. We are considering full replication, where copy of same database will be available at all sites. In order to ensure consistency there are several replica control schemes [4][5][6]. Distributed 2PL[4][5] is one of them. It is also a pessimistic approach in which data availability is sacrificed due to locking.

Revised Manuscript Received on July 02, 2019

Anupam Singh, **Raghuraj Suryavanshi**, Department of Computer Science and Engineering, PSIT Kanpur, India.

Divakar Singh Yadav, Department of Computer Science and Engineering, IET Lucknow, India.

In distributed 2PL, all sites are required to be available for the commitment of the transaction. In distributed environment it is very difficult to ensure the availability of all sites. To overcome this problem, we are considering majority based replica control protocol [4][5]. Majority based replica control protocol handles the faulty situations where group of sites are not available due to any failure reason. In consideration of faulty environment, we have introduced notion of resending of vote request, if majority is not achieved. In this regard, first coordinating site checks the availability of more than half of the sites, if majority is not fulfilled it resends the vote request to non responder sites. The frequency of resending of vote request depends on a constant value called as threshold. In order to give verification and formal development of our model, we are using Event-B that is event-driven formal method. Formal methods [7][8][9][10] are mathematical techniques that are used to verify the correctness of model. Event-B is a formal method which supports development of model in distributed environment. To ensure the correctness of model, proof obligations which are generated by Event-B model must be discharged. Rodin [11][12][13] is an eclipse based framework which provides an environment to write Event-B specifications and to discharge proof obligations. The remainder of this paper is organized as follows: Section 2 provides introduction of the B Method, section 3 provides system model informally, section 4 presents formal development of fault tolerant majority based protocol and section 5 concludes the paper.

II. EVENT-B FORMAL METHOD

Event-B [14][15][16][17][18] is the successor of the B method permitting to model discrete systems using mathematical notations. An Event-B specification is made of two elements: context and machine. Context presents static part and machine shows dynamic part of model. The machine contains variables, invariants and events. Events are checked by different conditions called as guards. When the guards of the event become true list of actions will be performed. The state variables are modified by set of events. The invariants state properties that are defined on variables must be always satisfied, when variable changes its value in different events.

III. INFORMAL DESCRIPTION OF FAULT-TOLERANT MAJORITY BASED PROTOCOL

In this section, we present an informal discussion on fault tolerant majority based protocol to control database replication. In full replicated database system, common data object are present at all sites.
Majority based concurrency

control protocol is appropriate choice to control replicas under network partitioning or site unavailability. This protocol ensures that any transaction will be committed under the availability of more than half of the sites. In order to check the availability, the transaction submission site i.e. coordinator site will send vote request message to all other sites (participating sites). Participating site will send the response to coordinating site. Coordinating site will count total number of responses from participating site. If total number of responses exceeds the majority (more than half of the sites) then the coordinator will check the latest copy of replica. To find out latest copy of replica, version number is used. Each time when writing is done on replica, version number will be incremented by one. Therefore, the site having largest version number will have latest copy of replica. After execution of transaction on latest copy of replica, the latest version number and replica will be sent to all available sites. In this paper, we have also considered the delayed response from participating site. For handling delayed response, we have introduced notion of timer at coordinating site. The coordinating site will activate timer when request message is broadcast to all other participating sites. The sender will wait for response for that time period. When time out occurs, coordinating site will count the total number of responses received. If the majority of sites are not available, then the sender assumes that either the site is not available or the message is delayed. The coordinating site resends the request message to those sites whose response is not available at coordinating site yet.

IV. FORMAL DEVELOPMENT OF FAULT-TOLERANT MAJORITY BASED PROTOCOL

In this section, we present formal modeling of fault tolerant majority based protocol. In the context part of the model the site , message, loc lock man and transaction are declared as carrier set. The set *status*, *vstatus* and *timeout* are declared as enumerated set. The *status* specifies the state of the transaction having values *abort, commit* and *pending* while *vstatus* having values *granted* and *notgranted.* The set *timeout* is enumerated set having values *active* and *expire*.
The description of variables which are declared in
machine part are as follows:(fig. 1)
- The variable vote_response represents the set of responses from participating site to coordinating site.
- The variable active_trans is a set of all active transactions submitted at any site.

- Variable *t_status* specifies status of transaction at any site. The mapping (*ss mtt) mabort : t_status* indicates that status of transaction at site *ss* is abort.
- The variable *vr_status* defines whether vote request granted or not from site to other site.
- The *lock_request* variable specifies lock request message from site to its local lock manager.
- The variable *lock_status* specifies status of lock request at any site. The mapping (*ss m granted) : lock_status* indicates that lock status of site *ss* is granted.
- The variable *vn* is a version number of a site which is a natural number.
- The variable *activesite* specifies all available sites that is participating in the processing of transaction.
- The variable *trans* defines the set of all active transactions.
- The variable *cosite* specifies set of coordinator site.
- Variable *sitetime* specifies status of timer at particular site. The time out value of any site may be either expire or active.
- The variable *verval* represents value of version number which is defined as set of natural number.
- The variable sender is defined as *sender : (message m site).* The mapping *(mmmss):sender* indicates that message mm has been sent by sender *ss*.
- Variable deliver ensures delivery of message at particular site.
- The variable *vnm* specifies the version number of message.
- The variable *msgfortran* specifies update message for any transaction. The mapping *(mm m tt) : msgfortran* indicates that update message mm for transaction *tt* has been sent.
- The variable *noofattempt* represents number of attempts a vote request can be sent from coordinating site to other participating sites.
- The variable *threshold* is a constant which defines the upper limit of number of attempts. It is used to fix up the number of times a site can resend vote request to participating sites if any failure occurs

The variables *vn, noofattempt* are initialized to zero while *threshold* is initialized to some positive constant. Remaining variables are initialized to ϕ.

The Event-B specification of our model are as follows:

4.1. Transaction initiation and lock request to its local lock manager

Submission of transaction is shown in fig.2(*Transaction_ Initiate* event). This event specifies the submission of fresh transaction. The site on which a new transaction is submitted, known as coordinator site. The guards (*grd3* and *grd4*) of this event shows that transaction *tt* is a fresh transaction and it is not active at site *si* respectively. Due to initiation of this event tt will become active at site *si* (*act1* and *act2*). The action *act3* make site si as coordinating site for transaction *tt*. The action act4 set the status of transaction as pending.

After submission of transaction, site sends lock request message to local lock manager for requested data items by the transaction (*Send Lock Req* event of fig.2). The guard *grd3* specifies that site *ss* has not done request for its local lock manager *llm*. The guard *grd4* ensures that transaction *tt* is active at site *ss.*

Due to occurrence of the event lock request set will make the entry of lock requests of the site *ss* to its local lock manager *llm (act1).*

4.2. Sending vote request to participating sites and vote response

This event (*Send_Vote_Req*) specifies sending of vote request message by coordinating site to all other sites(participating sites). In this model fault tolerance for site crash and delayed message lost has been considered.

```
MACHINE Majority
SEES context1
VARIABLES
activesite, ,vote_request, active_trans,
t_status,vr_status,lock_request, lock_status, vn, cosite,
sitetime, verval, trans, sender,deliver, vnm, msgfortran,
noofattempt, threshold

INVARIANTS
    inv1  :  vote_response∈site ⇸ site
    inv2 :   active_trans∈site↔transaction
    inv3  :  t_status ∈(site×transaction)⇸status
    inv4  :  vr_status ∈(site × site) ⇸ vstatus
    inv5  :  lock_request ∈site ↔ loc_lock_man
    inv6  :  lock_status ∈site → vstatus
    inv7  :  vn∈site→ℕ
    inv8  :  activesite⊆site
    inv9  :  finite(activesite)
    inv10 :  trans ⊆transaction
    inv11 :  cosite ∈trans→site
    inv12 :  sitetime ∈site→timeout
    inv13 :  verval⊆ℕ
    inv14 :  sender ∈message⇸site
    inv15 :  deliver ∈site↔message
    inv16 :  vnm ∈message⇸ℕ
    inv17 :  msgfortran ∈message⇸trans
    inv18 :  noofattempt ∈site→ℕ
    inv19 :  threshold ∈ℕ
```

Figure 1: Variables and Invariants of Machine

In faulty environment, it may possible that participant sites are available but the reply messages sent by them may be delayed. It may cause to abort transaction at coordinator site because of majority is not achieved although sufficient no of participant sites are available but their reply messages were delayed. In our approach, we are handling this situation by rebroadcasting request messages only to those participant sites from where response has not been received.

In this event (*Send_Vote_Request* of fig. 3), Site *ss* is a coordinating site for transaction tt that is ensured through guard *grd2*.The guards (*grd3* and *grd4*) ensures that tt is an active transaction at site *ss* and it’s request to its local lock manager is fulfilled respectively. The guard (*grd5*) specifies that more than half of the sites are not available. Due to occurrence of this event, status of the site *ss* for transaction *tt* will be set to pending and response timer is activated (*act1*) through action (*act2*).

```
Transaction_Initiate≙
ANY si, tt
WHERE
grd1   :si∈site
grd2   :tt∈transaction
grd3   :tt∉trans
grd4   :(si↦tt)∉active_trans
grd5   :(si↦tt)∉dom(t_status)
THEN
act1  : active_trans≔active_trans ∪
        {si↦tt}
act2  :trans≔trans∪{tt}
act3  :cosite(tt)≔si
act4  : t_status≔t_status∪
        {(si↦  tt)↦pending}
END
Send_Lock_Req  ≙
ANY  ss, llm, tt
WHERE
grd1  :  tt∈trans
grd2  :  llm ∈loc_lock_man
grd3  :  (ss↦llm) ∉lock_request
grd4  :  (ss ↦tt) ∈active_trans
THEN
act1  :   lock_request ≔lock_request ∪
           {ss↦llm}



END
```

Figure 2: Transaction Submission and Sending of Lock Request

The event (*Send Response*) specifies the sending of response from participating site to coordinating site (see fig. 3).

```
Send_Vote_Request ≙
ANY tt, ss, llm
 WHERE
grd1  :  tt ∈ trans
grd2  :  ss=cosite(tt)
grd3  :  (ss↦ tt) ∈ active_trans
grd4  :  ss↦llm ∈ lock_request
grd5  :  card(activesite)<card(site) ÷2
grd6  :  noofattempt < threshold
THEN
act1  :  t_status(ss↦tt) ≔ pending
act2  :  sitetime(ss)≔ active

act3  :  noofattempt(ss)≔ noofattempt(ss)+1

END

Send_Response ≙
ANY  sj,ss,tt
WHERE
grd1  :sj∈site
grd2  :tt∈trans
grd3  :ss = cosite(tt)
grd4  :(sj↦ss)∈dom(vr_status)
grd5  :(sj)∉dom(vote_response)
grd6  :vr_status(sj↦ss)=notgranted
grd7  :sitetime(ss)=active
grd8  :(ss↦tt)∈dom(t_status)
grd9  :t_status(ss↦tt)=pending
THEN
act1  :  vote_response ≔ vote_response
          ∪{sj↦ss}
END
```

Figure 3: Vote Request and Send Response

After receiving of vote request for a transaction *tt* from coordinating site, participating sites send response to it. The guard *grd6* specifies that participants site *sj* has not sent the vote response to coordinator site *ss*. The guard *grd7* ensures that response timer is active at coordinating site *ss* and status of transaction *tt* is pending is ensured by guard *grd9.*

Due to occurrence of this event vote response will be sent by participating site *sj* to coordinating site ss.

```
Message_Failure ≙
ANY sj, ss, tt
WHERE
grd1  :sj∈site
grd2  :tt∈trans
grd3  :ss=cosite(tt)
grd4  :sj≠ss
grd5  :(sj↦ss) ∈(vote_response)
THEN
act1  :vote_response ≔ vote_response \
        {sj↦ss}
END

Accept_VR  ≙
ANY sj, tt, ss, llm
WHERE
grd1  :  tt∈trans
grd2  :  ss=cosite(tt)
grd3  :  sj∉activesite
grd4  :  sitetime(ss)=active
grd5  :  (sj↦ss) ∈vote_response

grd6  :  (ss↦tt)∈active_trans

grd7  :  (sj↦tt)∉active_trans
grd8  :  (ss↦llm) ∈lock_request
THEN
act1  :  vr_status(sj↦ss)≔ granted
act2  :  activesite≔ activesite∪{sj}
act3  :  active_trans≔active_trans ∪
          {sj↦tt}
act4  :  verval≔verval∪{vn(sj)}
END
```

Figure 4: Message Failure and Acceptance of Vote Response

4.3. Failure of Message and Acceptance of Vote Response

This event (*Message_Failure event*) specifies modelling of delayed message (see fig. 4). If there is any situation when site is available but vote response is not received on coordinating site then the entry in vote response will be omitted from vote response. The guards *grd3* and *grd4* ensures that coordinating site and participating sites are different. The guard *grd5* ensures participant site *sj* has sent the vote response to coordinating site *ss*. This event removes the entry of vote response since message is delayed or lost.

The event (*Accept_VR*) models the acceptance of vote response at coordinating site. Site *ss* which is defined as coordinating site checks the vote responses from participating sites(*sj*). The guard *grd5* specifies that participating site *sj* has sent the response to coordinating site ss. The guard *grd7* ensure that in the knowledge of coordinating site transaction tt is not active at participating site *sj* since response has not received. Due to occurrence of this event the vr status will be updated by granted (*act1*). The set of active site will also have updated for transaction *tt (act2 and act3).*

4.4. Finding Maximum Version Number of Replicas and Commit Operation at Coordinating Site

This event models the computation of maximum version number to find the latest copy of replica (*Max_Version* of fig. 5). The replica which has highest version number will be latest one. Initially, version number of all sites will be zero and it will be incremented by one. The set *verval* stores version numbers (*vn*) of each site. The guard grd5 and *grd6* that value of variable *maxver* will be the maximum value of all versions present in *verval* set. The action *act1* assigns the maximum version number to site *ss*.

The event *Cord Commit* specifies commitment of transaction at coordinating site *ss* (see fig. 5). At coordinator site *ss(grd2)*, if majority is fulfilled then the transaction executes and change its state from pending to commit state. For commitment more than half of the sites(*grd4*) must be available. After the commitment version number will be incremented by one.

```
Max_Version ≙
ANY ss, maxver, tt
WHERE
grd1  :  maxver∈ℕ
grd2  :  tt∈trans
grd3  :  ss=cosite(tt)
grd4  :  sitetime(ss)=expire
grd5  :  ∀x·x∈verval⇒maxver≥x
grd6  :  maxver=max(verval∪{0})
THEN
act1  :  vn(ss)≔maxver
END

Cord_Commit ≙
ANY sj, tt, ss
WHERE
grd1  :  tt∈trans
grd2  :  ss=cosite(tt)
grd3  :  (sj↦tt)∈active_trans
grd4  :  card(activesite)>card(site)÷2
grd5  :  (sj↦ss)↦granted ∈vr_status
THEN
act1  :  vn(ss)≔vn(ss)+1
act2  :  t_status(ss↦tt)≔commit
END
```

Figure 5: Maximum version number and coordinator commit

4.5. Broadcast Updations to Other Participating Sites and Receive

This event (*Broadcast*) models the transfer of updations done on coordinating site to all other participating sites (see fig. 6). After commitment of transaction the coordinating site broadcast a message to all sites to maintain consistency in database. The guard *grd1* and *grd2* specifies that transaction *tt* is an active transaction whose coordinating site is *ss*. The Guards *grd3* and *grd4* ensures that message mm has not been sent. The guard *grd6* specifies that transaction has been committed at coordinator site *ss*. On occurrence of the event, message *mm* is broadcasted by site *ss (act1)* sender will be updated(*act1*) and version number of coordinating site has been assigned to message (*act2*). The action *act3* add the transaction *tt* to *msgfortran* set. This event(*Receive*) specifies

```
Broadcast
ANY ss, mm, tt
WHERE
grd1  :  tt∈trans
grd2  :  ss=cosite(tt)
grd3  :  mm∈message
grd4  :  mm∉dom(sender)
grd5  :  (ss↦tt)∈dom(t_status)
grd6  :  t_status(ss↦tt)=commit
THEN
act1  :  sender≔sender∪{mm↦ss}
act2  :  vnm(mm)≔vn(ss)
act3  :  msgfortran(mm)≔tt
END
Receive
ANY ss, mm, tt
WHERE
grd1  :  ss∈Site
grd2  :  mm∈dom(sender)
grd3  :  mm∈dom(vnm)
grd4  :  mm↦tt ∈msgfortran
THEN
 act1  :  deliver≔deliver∪{ss↦mm}
END
```

Figure 6: Broadcast and Receive Event

receiving of message at participating site regarding updations given by coordinator for transaction *tt* (see fig. 6). The guard *grd2* ensures that message mm has already been sent.The guard *grd3* checks the version number of the message(*vnm*). On occurrence of this event deliver set will be updated with the entry of message and its sender details (*act1*).

4.6. Commit Operation of Transaction on Participating Site and Abort on Coordinator Site

This event (*Part_Commit*) models the execution of commit operation at participant site (see fig. 7). For maintaining the consistency, all participating sites should execute commit operation. Transaction *tt* is an active transaction at site *ss* (*grd6*, *grd7*) then participating sites assigned latest version number.

The guard *grd10* checks the delivery of message while *grd11* verifies that whether the majority is achieved or not. Due to occurrence of the event, status of participating sites for transaction *tt* will be committed and new version number will be allotted (*act1*, *act2*).

```
Part_Commit
ANY ss, tt, si, mm
WHERE
grd1  :  ss∈activesite
grd2  :  tt∈trans
grd3  :  si=cosite(tt)
grd4  :  mm∈dom(sender)
grd5  :  mm∈dom(vnm)
grd6  :  (ss↦tt)∈active_trans
grd7  :  (ss↦tt)∈dom(t_status)
grd8  :  t_status(ss↦tt)=pending
grd9  :  sender(mm)=si
grd10 :  ss↦mm∈deliver
grd11 :  card(activesite)>card(site)÷2
THEN
act1  :  t_status(ss↦tt)≔commit
act2  :  vn(ss)≔vnm(mm)
END

Co_Abort
ANY sj, tt, ss
WHERE
grd1  :  tt∈trans
grd2  :  ss=cosite(tt)
grd3  :  sj∈activesite
grd4  :  tt∈transaction
grd5  :  (ss↦tt)∈active_trans
grd6  :  card(activesite)<card(site)÷2
grd7  :  (sj↦ss)∈dom(vr_status)
grd8  :  vr_status(sj↦ss)=granted
grd9  :  noofattempt(ss)=threshold

THEN
act1  :  t_status(sj↦tt)≔abort
END
```

Figure 7: Participant Commit and Coordinator Abort

When half of the sites are not available (*Cord_Abort* event) and number of attempts for revoting exceeds the threshold then the transaction will be aborted. In event (*Co_Abort* of figure 7), transaction *tt* has been submitted at coordinator site is ensured through guards *grd2* and *grd4*. The guard *grd6* specifies that majority of sites are not available. Participating sites (*sj*) are available in *activesite* which is ensured through *grd3*. The guard *grd8* specifies that the vote request status of participating site *sj* on coordinator site *ss* is granted. Due to occurrence of this event status of transaction *tt* will be aborted as majority is not fulfilled (*act1*).

4.7. Unlock and Timeout event

In figure 8, When a transaction on site changes its state from pending to either commit or abort, the site executes unlock request to its local lock manager (*Unlock Event*). The guards, *grd2* and *grd3* check whether lock is already acquired or not. If lock is acquired and transaction has been committed successfully then the entry for the acquired lock will be removed from lock request (*act1*). This event(*Timeout*) models timing boundaries for a transaction tt (see fig. 8). The guard *grd1* specifies that transaction *tt* is an active transaction submitted on site *ss* which is coordinator site (*grd2*). The guard *grd3* checks that the timer is active or not. On occurrence of the event, if duration is completed then it will be expired for the identified site

```
Unlock ≙
ANY ss, llm, tt
WHERE
grd1  :  ss∈site
grd2  :  llm∈loc_lock_man
grd3  :  (ss↦llm)∈lock_request
grd4  :  (ss↦tt)∈dom(t_status)
grd5  :  t_status(ss↦tt)=commit
THEN
act1  :  lock_request ≔ lock_request \
         {ss↦llm}
END

Timeout ≙
ANY ss, tt
WHERE
grd1  :  tt∈trans
grd2  :  ss=cosite(tt)
grd3  :  sitetime(ss)=active
THEN
act1  :  sitetime(ss)≔expire
END
```

Figure 8: Unlock and Timeout

V. Conclusion

In this paper, we have done the formal development of fault tolerant majority based replica control protocol. Formal methods are techniques to verify the correctness of the system mathematically. Replica control is challenging issue to provide consistency in distributed environment. In faulty environment network may be partitioned due to link failure or messages may be delayed. In order to control the replica under network partitioning, we are considering majority based replica control protocol. In this protocol even sites are available but due to delayed messages majority will not be fulfilled incurred extra cost for completion of transaction. In this paper, we have introduced the notion of time period which is allotted to coordinator site (transaction submission site). The coordinator site will resend the request messages to those participating sites from where messages are not received in allotted time period. We have also used a control variable named as threshold which control total number of times a request message can be sent by transaction coordinator site. For the formal verification of our model, we have considered Event-B as a formal method.

In this model, we have formally verified replica control protocol using Event-B. Event-B model generates proof obligations. In order to ensure correctness of model these proofs must be discharged. While discharging proofs all invariants are preserved (no violation) We have considered RODIN platform for writing B specifications.

In order to verify the correctness of our model we have also added following invariant.
!ss, tt(ss : SITE &tt : trans & ss = cosite (tt) & (ss m tt) : dom(t_status) & t_status (ss m tt) = commit) y card(activesite) > card(site) / 2)
This invariant ensures that site ss is coordinating site and status of transaction *tt* is commit then majority of sites are available. In our Model 74 proofs were generated by system, out of which 56 proofs are discharged automatically and 18 proofs are discharged interactively. While discharging the proofs of the model it gives clear insight about the protocol. In future we will extend our model for dynamic partitioning in distributed environment
through refinement.

REFERENCES

1. M. Ozsu and P. Valduriez: Principles of Distributed Database Systems.
2. Pearson Education (Singapore) Pte.Ltd. India 2004.
3. M. Singhal, N.G. Shivratri: Advanced Concepts in Operating Systems.
4. Tata McGraw- Hill Book Company, 2012.
5. R. Suryavanshi and D.Yadav,Rigorous Design of Lazy Replication System Using Event-B, Communications in Computer and Information Science ISSN: 1865-0929, Volume 0306, Springer, Verlag Germany 2012, pp 400-411.
6. Helal, A., Heddya, A. and Bhargava, B.: Replication Techniques in
7. Distributed System. Kluwener Academic Publishers (1997).
8. J. Gray, P. Helland, P. E. ONeil, and D. Shasha. The dangers of replication and a solution. In Proc. of the ACM SIGMOD Int. Conf. on Management of Data, pages 173-182, Montreal, Canada, June 1996.
9. Kemme, B., Alonso, G.: A new approach to developing and implementing eager database replication protocols. ACM Transaction Database System, 25(3), 2000, pp 333-379.
10. Paul Ammann, Sushil Jajodia, and Indrakshi Ray. Using formal methods
11. to reason about semantics-based decompositions of transactions. In
12. Umeshwar Dayal,Peter M. D. Gray, and Shojiro Nishio, editors, VLDB
13. . Morgan Kaufmann, 1995, pp 218-227.
14. D. Yadav and M. Butler. Application of Event B to global causal ordering for fault tolerant transactions. In Proc. of Workshop on Rigorous Engineering of Fault Tolerant System, REFT05,Newcastle upon Tyne, 19 July 2005, pp 93-103.
15. Paul Ammann, Sushil Jajodia, and Indrakshi Ray. Applying formal
16. methods to semantic-based decomposition of transactions. ACM Transaction on Database System., 22(2), 1997, pp 215-254.
17. Jean-Raymond Abrial and Dominique Cansell. Clickn prove: Interactive
18. proofs within set theory. In David A. Basin and Burkhart Wolff, editors, TPHOLs,volume 2758 of Lecture Notes in Computer Science, Springer, 2003, pp 124.
19. C Metayer, J R Abrial, and L Voison. Event-B language. RODIN deliverables ,http://rodin.cs.ncl.ac.uk/deliverables/D7.pdf, 2005.
20. M. Butler, J.-R. Abrial, and R. Banach, From Action Systems to Distributed Systems: The Renement Approach. Tayloramp; Francis,2016, ch. Modelling and Hybrid Systems in Event-B and Rodin.
21. Michael Butler, Cli B. Jones, Alexander Romanovsky, and Elena Troubitsyna, editors. Rigorous Development of Complex Fault-Tolerant Systems [FP6 IST-511599 RODIN project], volume 4157 of Lecture Notes in Computer Science. Springer, 2006.
22. D. Yadav and M. Butler. Formal specifications and verification of message ordering properties in a broadcast system using Event B. In Technical Report,School of Electronics and Computer Science, University of Southampton, Southampton, UK, May 2007.
23. R. Suryavanshi, D. Yadav, Modeling of Multiversion Concurrency
24. Control System Using Event-B in Federated Conference on Computer
25. Science and Information systems (FedCSIS) 9-12 September,
26. Poland,indexed and published by IEEE ISBN 978-83-60810-51-4, 2012, pp 1397-1401.
27. R. Banach, M. Butler, S. Qin, N. Verma, and H. Zhu,Core Hybrid Event-B I: Single Hybrid Event-B machines, Science of Computer Programming, 2015.
28. E. Elsayed , G. El-Sharawy and E.Sharawy, Integration Of Automatic
29. Theorem Provers In Event-B Patterns,International Journal of Software
30. Engineering amp; Applications (IJSEA), Vol.4, No.1, Jan. 2013.
31. D. Yadav and M. Butler. Formal specifications and verification of message ordering properties in a broadcast system using Event B. In Technical Report,School of Electronics and Computer Science, University of Southampton, Southampton, UK, May 2007.

AUTHORS PROFILE

Anupam Singh is pursuing Ph.D. in Computer Science and Engineering. His research area is formal verification and validation. Mr. Singh is reviewer of some reputed journals.

Dr. Raghuraj Suryavanshi has done phd in computer sciecnce & Engineering. He has published papers in the area of formal verification of critical properties of distributed system. He was also awarded as Teacher fellowship award from state university AKTU Lucknow. Mr. Singh has also visited many countries for their research presentation and special talk in the area of formal verification.

Dr. Divakar Singh Yadav has done Ph.D. (Computer Science), University of Southampton, United Kingdom. His area of research is Formal Methods, Distributed Computing, Database Systems, Verification of Transactional Information Systems, Rigorous Design of Distributed Transactions.