FORMAL SPECIFICATION AND VERIFICATION OF TOTAL ORDER BROADCAST THROUGH DESTINATION AGREEMENT USING EVENT-B

Arun Kumar Singh[1] and Divakar Yadav[2]

[1]Department of Electronics Engg. IET, Lucknow,India
[2]Department of Computer Science & Engg. IET,Lucknow,India

ABSTRACT

A reliable broadcast is communication primitive used to develop fault tolerant distributed applications. It in due course delivers messages to all participating sites irrespective of their ordering. Total order broadcast impose restriction on message ordering and satisfies total order requirement.

A clear specifications, rigorous validation and verification is key to obtain better design of dependable services in such applications. With the help of formal methods one can specify and verify systems in systematic rather than ad hoc manner. It reveals ambiguities, incompleteness, and inconsistencies in a system by facilitating clear specification, rigorous validation and verification.

In this paper, we present a formal development of total order broadcast. The model have been developed and checked by using event-B techniques supported by the RODIN tool. Event-B is a formal technique that supports the incremental design of a distributed applications using notion of refinements.

KEYWORDS

Total order broadcast, Event-B, reliable broadcast.

1. INTRODUCTION

The verification and specification of fault- tolerant distributed application are difficult due to unavoidable concurrency and absence of global clock [1]. In reliable broadcast no assumptions on time can be made, although it is ensured that messages will be delivered irrespective of their ordering at the sites.

The delivery ordering of messages in a distributed environment can be ensured in a better way by using group communication primitives.

One such primitive is total order broadcast. The total order broadcast and multicast is an important problem for fault tolerant distributed application.

The *total order* [2] broadcast is a primitive for group communication which ensures that a message is delivered to all the recipients in the same order which may not be the same order in which the messages were sent. In this context, Hadzilacos and Toueg [3] defines the Reliable Broadcast as a broadcast that posses the characteristics of Validity, Agreement and Integrity. A broadcast is supposed to posses the *Validity characteristics if* a correct process broadcasts a message *m*, and it eventually delivers *m*. A broadcast is supposed to be in *Agreement if* all correct

processes eventually delivers the same set of messages. *Integrity ensures that* spurious messages are never delivered. That is every correct process delivers any message *m* at most once and that too if *m* was previously broadcast by *sender* (*m*). Few additional requirements are imposed on the order of delivery of messages in some variants of Reliable broadcast. [2].

Depending on group communication primitives, a large numbers of broadcast protocols have been developed. Lots of literature available on fault tolerant distributed applications where different group communication services are used but the use of formal methods in such applications are rare except [4] and few where it is applied to verify the properties of algorithms [5, 6].

Based on ordering mechanisms, total order broadcast have been categorized as communication history, privilege-based, moving sequencer, fixed sequencer and destinations agreement.

Although a number of algorithms have been devised in each of these categories, a very limited attempt [2] has been made to develop the specification and prove the correctness of these algorithms. Formalism helps to get a clear specification and a sound proof of correctness of the algorithms and thus helps in understanding the limits within which an algorithm can be used.
In this paper, we develop a formal specification of the destination agreement total order broadcast which is represented by many important algorithms such as Skeen [7], Chandra and Toueg [8], ATR [9], SCALATOM [10] etc. In these algorithms, an agreement between destination processes decides the delivery order.

Section 2 of the paper presents necessary background and an informal discussion of various ordering properties is given. In section 3, we introduce the basic notations of event-B, the techniques chosen for formal specification of the system. In section 4, we describe the destination agreement algorithm for which formal specification has been developed in section 5. Section 6 concludes the paper.

2. BACKGROUND

A reliable broadcast can be used to deliver messages to the processes following a *FIFO Order*, *Local Order*, *Causal Order* or a *Total Order* providing higher ordering guarantees on the message delivery [2].

Ordering properties have been thoroughly discussed in [2, 11, 12, 13] and can be broadly classified as follows:

FIFO Order- In reliable broadcast messages are following FIFO Order, if the broadcast of a message M1 is preceded by a message M2 by a particular process than M1 will be delivered before M2 by each recipient process.

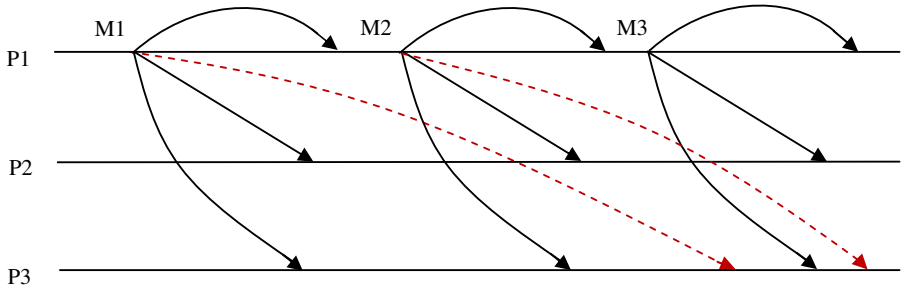

Fig 1: FIFO Order

As shown in figure1, the messages M1 broadcast by process P1 is delivered before M2 by each recipient processes and similarly M2 broadcast by same process P1 is delivered before M3 by each recipient processes are said to be in FIFO Order. The violation of FIFO Order is shown in the figure 1 by dotted lines.

Local Order-In reliable broadcast, if the delivery of a message M1 is preceded by broadcasting the message M2 by a particular process, than M1 will be delivered before M2 by each recipient process than messages are said to be in Local Order.

In the figure 2, the messages M1, M2 and M3 are said to be in local order as process P3 before broadcasting message M3 delivers message M2 and process P2 delivers M1 before broadcasting M2.The local order is violated by delayed messages M1 and M2 as depicted using dotted line.

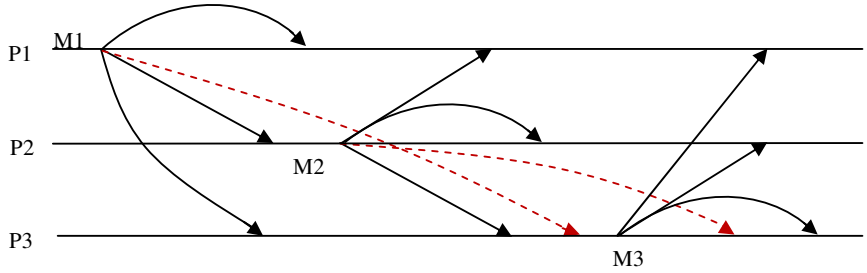

Fig 2: Local Order

Causal Order: In reliable broadcast messages are following Causal Order, if the broadcast of a message M2 is causally preceded by the broadcasting of message M1than each process will deliver M1 before delivering M2.

The causal precedence relation is expressed by the symbol → . The relation e→*f* signifies that *e* causally precedes *f,* where *e* and *f* are two events of distributed system. The *causal precedence is an irreflexive partial order* on the set of events. This relationship is also extended on the set of messages and defines the casual precedence among the messages in similar manner.

A casual precedence between message m_1 *and m2 hold*s whenever the casual precedence exists between broadcast events of m_1 *and m2 or casual precedence exists between* receive events of m_1 *and m2.*

The causal order broadcast delivers the messages respecting their causal precedence and is hybrid of FIFO and local order [2].

Total Order- In reliable broadcast messages are said to follow Total Order if the messages M1 and M2 both are delivered by the processes P1 and P2 then M1 is delivered before M2 by P1 if only if M1 is delivered before M2 also by P2.

Since delivery of messages in total order is not conforming to any particular order, it may or may not be satisfying causal relations. In the figure 3,the same sequence of messages M1,M2,M3 and M4 are delivered by all the processes P1,P2,P3 and P4 following the total order, also it conforms to causal order.

In the figure 4,the same sequence of messages M1,M3,M4 and M2 are delivered by all the processes P1,P2,P3 and P4 following the total order but nonconformity to causal order. Since the broadcast of message M3 is causally preceded by the broadcasting of message M2 therefore each

processes will have to deliver M2 before it delivering the M3 but here causal order is violated as M2 is delivered after M3 by each recipient process.

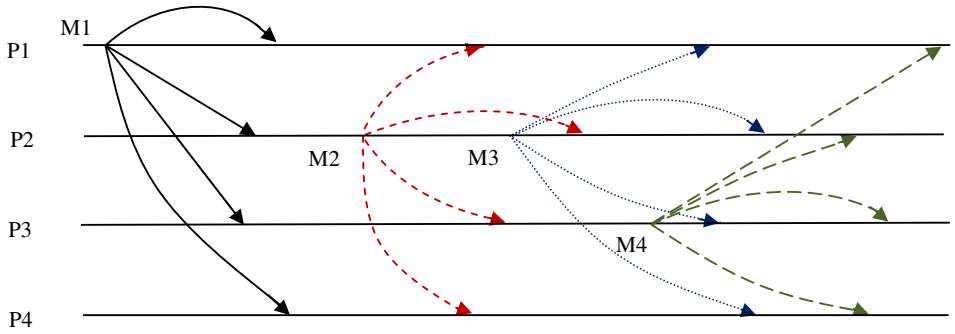

Fig 3: Total Order and Causal order

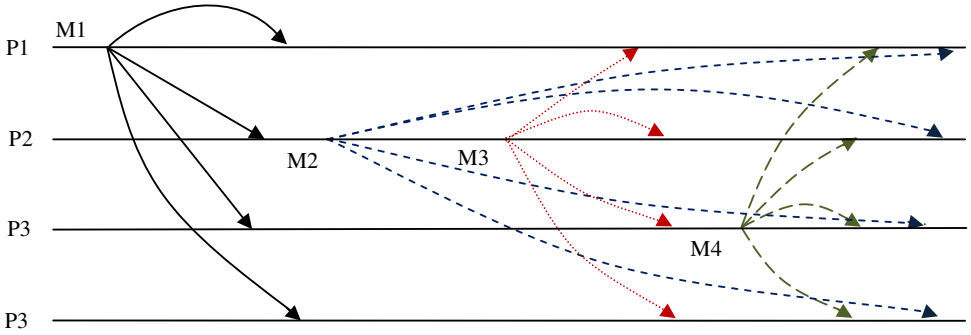

Fig 4: Total Order but not a Causal order

3. FORMAL MODELING USING EVENT-B

The basic notion of formal developments in Event-B [14, 15, 16] is that of model. Event-B has been derived from B-method [17] by incorporating the ideas of action systems [18] and is used for discrete-level modeling [19, 20]. The technique of abstraction and refinement is the basis of incremental design of systems in Event-B and is used for their specification and verification.
In Event-B, system is defined by state variables. The variables are modified by the events consist of guarded actions. The guard (G) of the event is expressed as a first order predicate. It represents the necessary conditions for the event to occur. The invariants are predicates on the state variables that represent system behavioral properties and have to be maintained by the activation of events. The actions of events are specified as simultaneous assignments of state variables. Events occur spontaneously and automatically whenever their guards hold true.

Refinement helps us to build a model in incremental fashion and allows us to obtain more concrete model from the abstract model by adding more functionality later. A refined model is thus one which is spatially larger than its abstraction because new variables are now able to be modified by some transitions, which could not have been present in the abstractions because of the reason that the variables in the refined model did not exist in the abstract model. This is realized by means of new events and the new variables. In refinement steps guards may be strengthened, new events may be introduced and variables may be added or removed. Abstract and concrete variables are related through *gluing invariants*. This requires the proof obligations for consistency checking and refinement checking to be discharged. Consistency checking

signifies that a machine preserves the invariants while refinement checking involves showing that gluing invariants are maintained. This ensures that behavior of a refined machine is in consistent with the abstract machine.

Event-B notation is based on set theory and most of it is self-explanatory. Some of the frequently used notations in our model is given in Table 1.

Table 1. Some B notations used frequently in system modeling

B Symbol	Description
1	Relational constructor
3	Total function
2	Partial function
*	Cartesian product
m	Mapping
N	Non zero natural number

4. INFORMAL DESCRIPTION OF TOTAL ORDER THROUGH DESTINATION AGREEMENT

The destination agreement algorithm achieves delivery order result from an agreement between destination processes [2] as shown in figure 5. An agreement is arrived by the destination processes on the basis of a unique sequence number. The algorithm is a modification of Skeen's algorithm which makes inference about the global timestamp in a decentralized manner [7].

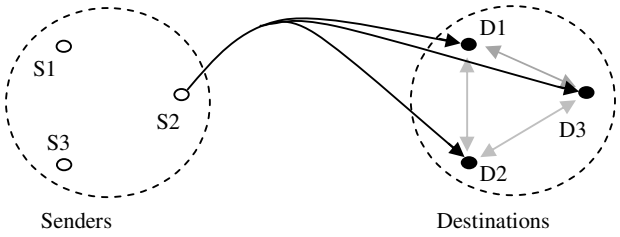

Fig 5: Total Order through destination agreement

In brief, the algorithm works as follows:

To broadcast a message m, a sender initiates the process by sending m to all destinations. The destination which receives m, assigns it a local timestamp and thereafter, this timestamp is sent to all the destinations. When a destination process has received all local timestamps for the message m from all destinations, maximum value from among all local timestamps is calculated. This maximum value of all local time stamps is defined as a unique global timestamp sn (m) assigned to m.

Messages are considered to be delivered only when it has been assigned its global timestamp. Delivery order of messages is in order of its global time stamps. In case more than one messages have same global timestamps, the tie is broken on the basis of sender identity [2].

5. SYSTEM MODEL

We start with a variant of destination agreement algorithm where agreement takes place through message sequence number. In the context of our model SITE and MESSAGE are declared as carrier set. The variables and invariants of machine are given in figure 6.

MACHINE Total_M
SEES Total_C
VARIABLES
sender, totalorder, delorder, tempdeliver, localtss, localtsm, rbm, rebroadcastmsg, totalbroadcastsite, finaldeliver, globaltsm
INVARIANTS
inv1 : $sender \in MESSAGE \nrightarrow SITE$
inv2 : $totalorder \in MESSAGE \leftrightarrow MESSAGE$
inv3 : $tempdeliver \in SITE \leftrightarrow MESSAGE$
inv4 : $localtss \in SITE \rightarrow \mathbb{N}$
inv5 : $localtsm \in (MESSAGE) \leftrightarrow \mathbb{N}$
inv6 : $rbm \in SITE \leftrightarrow (MESSAGE \nrightarrow MESSAGE)$
inv7 : $reebroadcastmsg \in MESSAGE \nrightarrow SITE$
inv8 : $totalbroadcastsite \in (SITE \times MESSAGE) \rightarrow \mathbb{N}$
inv9 : $finaldeliver \in SITE \leftrightarrow MESSAGE$
inv10 : $globaltsm \in MESSAGE \nrightarrow \mathbb{N}$

Fig. 6. Variables and Invariants of Machine

The descriptions of variables are as follows:

(i)The variable sender is a partial function from *MESSAGE* to *SITE*. A mapping of form *(mmmss):sender* indicates that message *mm* has been sent by site *ss.*
(ii)The variable totalorder represents a relation. It is written as: *totalorder* $\in MESSAGE \leftrightarrow MESSAGE$. It is equivalent to *pow(MESSAGExMESSAGE).* A mapping *M1mM2: totalorder* represents that message *M1* is totally ordered before message *M2*.
(iii)The variable *tempdeliver* specifies temporary delivery of messages at any site.
(iv)The variable *localtss* represents local timestamp of site. Similarly, variable *localtsm* indicates local timestamp of message. It is a relation which tells that message may have several timestamps. The main reason is that when any message *mm* is received at any destination site then that site increments its own timestamp and that timestamp will be assigned to message *mm*. Similarly, other sites also assign timestamp to message *mm*.
(v)In order to decide global timestamp of received message *mm* site will broadcast local check point number message *m* corresponding to message *mm*. The variable *rbm* records all such type of messages sent by site. The variable *rebroadcastmsg* maintains the information that local message has been sent by site.
(vi) The variable *totalbroadcastsite* counts total number of sites that have broadcast local checkpoint number message corresponding to received message *mm* for which global number

has to be completed. A mapping *totalbroadcastsite(ssmmm)=k* indicates that *k* site has sent local checkpoint number message corresponding to message *mm*.
(vii) The variable finaldeliver makes final delivery of messages at destination site. The variable globaltsm is used to assign final global timestamp to message.

INITIALISATION ≙
BEGIN
act1 : $sender := \emptyset$
act2 : $tempdeliver := \emptyset$
act3 : $localtss := SITE \times \{0\}$
act4 : $localtsm := \emptyset$
act5 : $rbm := \emptyset$
act6 : $reebroadcastMSG := \emptyset$
act7 : $totalbroadcastsite := (SITE \times MESSAGE) \times \{0\}$
act8 : $totalorder := \emptyset$
act9 : $finaldeliver := \emptyset$
act10 : $globaltsm := MESSAGE \times \{0\}$
END

Fig 7: Initialisation of total order machine

5.1 Broadcast of Message

Broadcast event is given in figure 8. This event formalizes the broadcasting of message from sender site. The message *mm* is fresh and has not been sent is ensured by guard *grd3.* The action *act1* of this event specifies broadcasting of message *mm* by site *ss.*

Broadcast ≙
ANY *ss, mm*
WHERE
grd1 : $ss \in SITE$
grd2 : $mm \in MESSAGE$
grd3 : $mm \notin dom(sender)$
THEN
act1 : $sender := sender \cup \{mm \mapsto ss\}$
END

Fig 8: Broadcast event

5.2 Temporary Delivery of Message

This event models the temporary delivery of message (figure 9). The message *mm* has been sent but not delivered at site *ss* is ensured through guard *grd3* and *grd4* respectively. Each time when a message is delivered, site increment its own time stamp by one *(act1)* and assign this time stamp to message *mm (act2).* The action *act3* makes the temporary delivery of message *mm* at site *ss.*

Temp_Deliver ≙
ANY *ss, mm*
WHERE
grd1 : $ss \in SITE$
grd2 : $mm \in MESSAGE$
grd3 : $mm \in dom(sender)$
grd4 : $(ss \mapsto mm) \notin tempdeliver$
THEN
act1 : $localtss(ss) := localtss(ss)+1$
act2 : $localtsm(mm) := localtss(ss)$
act3 : $tempdeliver := tempdeliver \cup \{ss \mapsto mm\}$
END

Fig 9: Temp_Deliver event

5.3 Re-Broadcasting of Message

In order to decide global time stamp, every site broadcast its own local time stamp to other site (figure 10). These event broadcasts a message *m* correspond to temporary delivered message *mm*. The message *mm* at site *ss* has been temporary delivered is ensured through guard *grd4*. The message *m* has not been broadcast is specified through guard *grd6*. This event rebroadcast message *m* corresponds to message *mm (act1).* The action *act2* assigns the timestamp of site to timestamp of message *m*. The action *act3* adds the message *m* in rebroadcast message list.

Re-Broadcast ≙
ANY *ss, mm, m*
WHERE
grd1 : $ss \in SITE$
grd2 : $mm \in MESSAGE$
grd3 : $mm \in dom(localtsm)$
grd4 : $ss \mapsto mm \in tempdeliver$
grd5 : $m \in MESSAGE$
grd6 : $m \notin dom(reebroadcastMSG)$
THEN
act1 : $rbm := rbm \cup \{ss \mapsto \{m \mapsto mm\}\}$
act2 : $localtsm(m) := localtss(ss)$
act3 : $reebroadcastMSG := reebroadcastMSG \cup \{m \mapsto ss\}$
END

Fig 10: Rebroadcast event

5.4 Delivery of Re-Broadcast Message

This event makes the delivery of rebroadcast message m correspond to message mm (figure 11). The guard *grd3* ensures that site *ss* has sent the message *m* corresponds to message *mm*. The message *m* has not been delivered at site *s* is ensured through guard *grd4*. This event makes the delivery of message *m* at site *s (act1)*. Site also counts the total number of broadcast site. Each time when a rebroadcast message corresponds to message *mm* is delivered count value is

incremented by one *(act2)*. The action *act3* adds the local time stamp of sending site as time stamp of message *mm*.

Rebroadcast_Delivery ≜
ANY *m, s, mm, ss*
WHERE
grd1 : $m \mapsto ss \in reebroadcastMSG$
grd2 : $mm \in dom(sender)$
grd3 : $ss \mapsto \{m \mapsto mm\} \in rbm$
grd4 : $m \notin tempdeliver[\{s\}]$
THEN
act1 : $tempdeliver := tempdeliver \cup \{s \mapsto m\}$
act2 : $totalbroadcastsite(s \mapsto mm) := totalbroadcastsite(s \mapsto mm)+1$
act3 : $localtsm := localtsm \cup \{mm \mapsto localtss(ss)\}$
END

5.5 Evaluation of Global Timestamp Message

This event formalizes computation of global timestamp (fig. 12). Temporary delivery of message *mm* has been done at site *ss* is ensured through guard *grd2*. The guard *grd3* ensures that site *ss* has received rebroadcast message from every site.

Evalute_GTS ≜
ANY *ss, mm, globalts, alltsm*
WHERE
grd1 : $ss \in SITE$
grd2 : $mm \in tempdeliver[\{ss\}]$
grd3 : $totalbroadcastsite(ss \mapsto mm) = card(SITE)$
grd4 : $mm \in dom(localtsm)$
grd5 : $alltsm = localtsm[\{mm\}]$
grd6 : $finite(alltsm)$
grd7 : $globalts = max(alltsm)$
$\forall m \cdot (m \in MESSAGE \wedge m \in dom(globaltsm)$
grd8 : $\wedge\ globaltsm(m) < globalts$ G
$(ss \mapsto m) \in finaldeliver)$
THEN
act1 : $globaltsm(mm) := globalts$
act2 : $localtss(ss) := globalts$
act3 : $finaldeliver := finaldeliver \cup \{ss \mapsto mm\}$
act4 : $totalorder := totalorder \cup (ran(finaldeliver) \times \{mm\})$
END

Fig 12: Evaluate_GTS event

The variable *alltsm* represents a relational image of relation *localtsm*. It contains all local timestamp of message *mm* received from other sites. The guard *grd7* select the maximum value of *localtimestamp*. The guard *grd8* ensures that all messages *m* whose global timestamp is less than selected max value for message *mm* will be already been delivered. Due to occurrence of this event maximum value of timestamp *globalts* will be assigned as global timestamp of message *mm (act1).* The site also update its local timestamp as *globalts(act2)*. The action *act3* makes the final delivery of message *mm* at site *ss*. The action *act4* specifies total order delivery of message *mm*.

6. CONCLUSION

Total order is a broadcast primitive which is used to ensure reliable delivery of messages. Destinations agreement algorithms which is one of the total order broadcast primitive, decides the order of delivery of messages on the basis of an agreement between destination sites. The categories of three different variants of agreement are agreement through message sequence number, agreement on a message set and agreement on the acceptance of a proposed message order.

In this paper, we have developed formal specifications of total ordering by destination agreement through message sequence number. We have used Event-B as formal method for writing the specifications and ensuring correctness of our model. The RODIN tool which provides complete framework for development of event-B models has been used for generation and discharge of proof obligation of the system. RODIN is eclipse based IDE which is used to develop formal specification of distributed systems.

There has not been any invariant violation signifying that model is consistent and critical behavioural properties of the system is maintained. One of the important invariant given below was added which ensures that all those messages whose ordering is done must be delivered:
$\forall m \cdot (m \in (dom(totalorder) \cup ran(totalorder)) \Rightarrow m \in ran(finaldeliver))$

A total of 27 proof obligations were generated by system. 23 of these were discharged automatically while 4 required interaction with the system.

REFERENCES

[1] Leslie Lamport and Nancy A. Lynch. Distributed computing: Models and methods. In *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*, pages 1157-1199. 1990.

[2] Xavier Defago, Andre Schiper, and Peter Urban. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Comput. Surv.*, 36(4):372-421, 2004.

[3] V. Hadzilacos and S.Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical Report TR 94 -1425, Cornell University, NY, 1994.

[4] Divakar Yadav and Michael 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, http://eprints.ecs.soton.ac.uk/14001/.

[5] Alan Fekete, Nancy A. Lynch, and Alexander A. Shvartsman. Specifying and using a partitionable group communication service. *ACM Trans. Comput.Syst.*, 19(2):171-216, 2001.

[6] C. Toinard, Gerard Florin, and C. Carrez. A formal method to prove ordering properties of multicast systems. *ACM Operating Systems Review*, 33(4):75-89, 1999.

[7] Birman,K.P. and Joseph, T. A., Reliable communication in the presence of failures. ACM Trans. Comput. Syst. 5(1): 47–76, 1987.

[8] Chandra, T. D. and Toueg, S. Unreliable failure detectors for reliable distributed systems. *J. ACM 43,* 2, 225–267, 1996.

[9] Delporte-Gallet, C. and Fauconnier, H. Real-time fault tolerant atomic broadcast. In *Proc. 18th Symp. on Reliable Distributed Systems (SRDS)*. Lausanne, Switzerland, 48–55, 1999.
[10] Rodrigues, L., Guerraoui, R., and Schiper, A. Scalable atomic multicast.In *Proc. 7th IEEE Intl. Conf. on Computer Communications and Networks*. Lafayette (LA), USA, 840–847, 1998.
[11] Kenneth P. Birman, Andr´e Schiper, and Pat Stephenson. Lightweight causal and atomic group multicast. *ACM Trans. Comput. Syst.*, 9(3):272–314, 1991.
[12] Roberto Baldoni, Stefano Cimmino, and Carlo Marchetti. Total order communications: A practical analysis. In Mario Dal Cin, Mohamed Kaaniche, and Andra´s Pataricza, editors, *EDCC*, volume 3463 of *Lecture Notes in Computer Science*, pages 38-54. Springer, 2005.
[13] Carlo Marchetti Stefano Cimmino and Roberto Baldoni. A classification of total order specifications and its application to fixed sequencer-based implementations. *Journal of Parallel and Distributed Computing*, 66(1):108-127, 2006.
[14] Abrial, J.R.: Extending B without Changing it (for developing distributed systems). Proc. of the 1st Conf. on the B method, H. Habrias (editor), France, pages 169–190, 1996.
[15] Jean-Raymond Abrial and Stefan Hallerstede. Refinement, Decomposition and Instantiationof Discrete Models: Application to Event-B. Fundamentae Informatica, 77(1-2), 2007.
[16] Abrial, J.R.:Modeling in Event-B: System and Software Design. Cambridge University Press, 2010.
[17] Jean-Raymond Abrial. The B-Book: Assigning Programs to Meanings. Cambridge University Press, 1996.
[18] Ralph-Johan Back. Refinement Calculus II: Parallel and Reactive Programs. In J. W.deBakker, W. P. deRoever, and G. Rozenberg, editors, Stepwise Refinement of Distributed Systems, volume 430 of Lecture Notes in Computer Science, pages 67–93, Mook, The Netherlands, May 1989. Springer-Verlag.
[19] Butler, M.: Incremental Design of Distributed Systems with Event-B, Marktoberdorf Summer School 2008 Lecture Notes (2008), http://eprints.ecs.soton.ac.uk/16910
[20] Butler, M., Yadav, D.: An incremental development of mondex system in Event-B.Formal Aspects of Computing 20(1), 61–77 (2008).

AUTHORS

Arun Kumar Singh is research scholar at U.P. Technical University. He holds a M. Tech. Degree in Electronics Engineering. His research interests include formal verification and distributed systems.

Divakar Yadav is professor at Department of Computer Science and Engineering at Institute of Engineering and Technology, Lucknow. He holds a Ph. D. Degree in Computer science from University of Southampton, UK. His research interests include distributed computing, databases and formal methods.