Rigorous Design of Partition-Aware Total Order Broadcast System using Event-B

Raghuraj Suryavanshi
Institute of Engineering and Technology
GBTU, Lucknow, India

Divakar Yadav
Institute of Engineering and Technology
GBTU, Lucknow, India

ABSTRACT
In distributed system, the sites communicate with each other by exchange of messages. A total order broadcast is a reliable broadcast that ensures delivery of messages to the sites in the same order. For the fault tolerant applications, ordering algorithm must support the network partitions and site failure. In this paper, we present a formal model of total order broadcast system in the presence of network partition and site failures. We have used Event-B as a formal technique for the development of our model. In this technique system is built in several stages by gradually introducing the details in the refinement steps. We outline an abstract model specifying total order broadcast and introducing the details in the refinement level for considering the case of network partitioning and site failure.

General Terms
Group communication primitives, B method, network partition.

Keywords
Total order broadcast, Formal methods, Event-B, sequencer.

1. INTRODUCTION
A distributed system is a collection of autonomous computer systems that cooperate with each other for successful completion of a distributed computation. Due to absence of common global clock or shared memory these systems communicate each other by exchange of messages which are delivered to them after arbitrary time delays. In such systems up-to-date knowledge of the system is not known to any process or site. This problem can be solved by group communication primitives that provide ordering guarantees on the delivery of messages. The group communication primitives have been proposed as a mechanism for the development of reliable fault-tolerant distributed applications [1]. A total order [1, 2] broadcast is one such primitive that guarantees the delivery of messages to the sites in the same order. For the fault tolerant applications, ordering algorithm must support the network partitions and site failure.

A considerable amount of effort is required to make distributed applications robust in the face of typical site and communication failure. In this paper, we present a formal model of partition aware total order broadcast system using Event-B. Event-B is a formal technique used for specifying and reasoning about complex systems. At the abstract level, we model the total order delivery of messages. In the refinement model we have considered, the case of network partitioning where the sites are continuously detach from the main group named as master group.

2. BACKGROUND
We have presented formal development of Byzantine immune total order broadcast system using Event-B in [3]. Our system model consist a set of sites. Any site can work as a sequencer and responsible for constructing total order. Total order broadcast is reliable broadcast that ensures that delivery order of the messages will be same at all the recipient sites.

The verification of different broadcast primitives and algorithm plays important role in distributed environment. Formal Methods provide systematic approach for building and studying the model. The most effective means to avoid system failures during a system operation is to eliminate or reduce design errors during design and development of system not afterwards when system complexity becomes overwhelming. Advantages and disadvantages of formal methods in industrial practice may be found in [4,5,6].

2.1 Event-B as a Formal Method
The B Method [7,8,9,10,11] is a model oriented state based method developed by Abrial. It represent the complete mathematical development of a Discrete Transition System. It is made of several components of two kinds: machines and contexts. Machines represent the dynamic part of model. This part is used to provide behavioural properties of model. It contains the variables, invariants, theorems, and events of a project. Contexts contain the static part of model. It contains carrier sets, constants, axioms, theorems.

Event-B [12,13,14,15] is event driven approach used to develop formal models of distributed systems through a series of refinement steps. An event is made up of three elements its name, guards and actions. The guards are the necessary conditions for the event to occur. An event known as initialization event has no guard and it gives initial position of the model. New variables, invariant, event may also add in the refinement step. The main purpose of adding new invariant and event is to find out more concrete specification.

2.2 Total Order Broadcast
Due to absence of common global clock or shared memory, the up-to-date knowledge about the system is not known to any site or process. These systems communicate through message passing where these messages are delivered after arbitrary time delays. This problem can be solved by group communication or broadcast primitives that provide ordering guarantees on the delivery of messages. Various definitions of the ordering properties have been discussed in [16, 17, 18, 19]. Total order is

one such primitive that ensures that delivery order of messages at all the sites will be same. A total order broadcast can be defined as a reliable broadcast which satisfies following requirement.

If processes p and q both deliver messages m1 and m2, then q delivers m1before m2 if and only if p delivers m1 before m2.

2.3 Partition aware Total order Broadcast system

Our system model contains a set of sites named as master group. When any site enters into the system it adds into the master group. The variable *groupid* is used to assign the id to a particular group. There are two types of operations attachment and detachment. During detachment, a group of sites detached from the master group and adds into the partitioned group. In the attachment process a group of sites from the partitioned group rejoins the master group and update their status by total order delivery of messages through sequencer present in master group. When the partitioning occurs in master group, each group assigned a different group id. Partitioned group contains the disjoint set of sites having different group id's which are detached from master group. In the partitioned group, the group having larger group id as comparison to the others group indicate that this group is detached after the others one. The new detached group may have some more messages which are not present at the others group which are partitioned before it. At some time new partitioned group (having larger group id) may be in contact with other partitioned group (having lower group id). The older partitioned group change its status from the newer ones by delivery of those messages which are not present there.

3. ABSTRACT MODEL

In the abstract model, we have designed total order delivery of messages. Any site can be chosen as a sequencer and this sequencer is used to ensure total order delivery of messages to participating sites. The SITE and MESSAGE is defined as carrier set. The additional machine variables are *sender*, *totalorder* and *tdeliver* (see Fig. 1).The sender is defined as partial function from MESSAGE to SITE. The mapping (mms)∶ sender indicates that message m has sent from the site s. The variable *totalorder* is defined as:

totalorder∶ MESSAGE1MESSAGE

Where MESSAGE1MESSAGE indicates set of relations between MESSAGE and MESSAGE. A mapping (m1mm2)∶ totalorder indicate that message m1 is totally order before m2. The variable tdeliver show that the message is delivered following a total order. A mapping (smm)∶ tdeliver represent that site s has delivered m following a total order.

3.1 Site Admit Event

This event (Fig. 2) adds a new site into the master group if it does not previously exist in the system. The site is not previously existed site is ensured by guard *grd2*.

3.2 Message Broadcasting (Broadcast event)

The specifications of the event Broadcast are given in Fig. 3. As outlined in the specification a site *ss* broadcast a message mm. The guard *grd4* of this event ensures that a message has not been previously sent by the sender.

3.3 Ordering of messages (Order event)

The Order event, given in Fig. 4. models the ordering of messages sent by the senders. The message has been sent is specified through *grd 4*. After receiving the messages the sequencer builds the total order and broadcast the messages to all sites. The guard *grd6* specifies that this message has not been delivered to a sequencer. The actions of this event construct a total order on a message and it is delivered to a sequencer.

3.4 Total Order delivery (ToDeliver event)

The event ToDeliver (see Fig. 5.) models the delivery of message *mm* to the site *ss* following the total order. The guard *grd5* ensures that the message *mm* has been sent by the sender and *grd6* ensures that it has been delivered to at least one site and it also implies that the total order on the message *mm* has been constructed.

```
MACHINE     Total_M
VARIABLES
        sender, totalorder, tdeliver, mastergroup
    INVARIANTS
        inv1  :  sender ∈ MESSAGE2SITE
        inv2  :  totalorder ∈ MESSAGE↔MESSAGE
        inv3  :  tdeliver ∈ SITE↔MESSAGE
        inv4  :    mastergroup ∈ ℙ(SITE)
INITIALISATION  ≙
    BEGIN
            act1 :  sender ≔ ∅       act2 : totalorder ≔ ∅
            act3 :  tdeliver ≔ ∅   act4:  mastergroup ≔ ∅
    END
```

Fig 1: Machine variables, invariants and initialization of abstract model

```
Site_Admit  ≙
    ANY  ss
    WHERE
            grd1  :  ss ∈ SITE     grd2:  ss∉ mastergroup
    THEN
            act1  :  mastergroup≔ mastergroup ∪ {ss}
    END
```

Fig 2: specification of site_admit event

```
Broadcast  ≙
    ANY  ss, mm
    WHERE
            grd1  :  ss ∈ SITE     grd2  : ss∈ mastergroup
            grd3  :  mm ∈ MESSAGE
            grd4  :  mm∉ dom(sender)
    THEN
            act1  :  sender≔sender∪{mm↦ss}
    END
```

Fig 3: Specification of Broadcast event

```
Order  ≙
    ANY
            ss, mm
```

```
WHERE
    grd1  :  ss ∈ SITE   grd2:  ss∈ mastergroup
    grd3  :  mm ∈ MESSAGE
    grd4  :  mm∈dom(sender)
    grd5  :  ss=sequencer
    grd6  :  (sequencer↦mm)∉tdeliver
THEN
    act1  :  tdeliver≔tdeliver∪{ss↦mm}
             totalorder≔totalorder ∪
    act2  :     (tdeliver[{sequencer}]×{m
             m})
END
```

Fig 4: specification of Order event

```
ToDeliver ≙
  ANY ss, mm
  WHERE
    grd1  :  ss ∈ SITE
    grd2  :  ss∈ mastergroup
    grd3  :  mm ∈ MESSAGE
    grd4  :  ss≠sequencer
    grd5  :  mm ∈ dom(sender)
    grd6  :  mm ∈ ran(tdeliver)
    grd7  :  ss↦mm ∉tdeliver
    grd8  :  ∀m·(m∈ MESSAGE ∧(m↦mm)
             ∈totalorder ⇒(ss↦m)∈tdeliver)
  THEN
    act1  :  tdeliver≔tdeliver∪{ss↦mm}
  END
```

Fig 5: Specification of ToDeliver event

4. REFINEMENT MODEL

In the refinement model, we add the specification of partitioning of sites where a group of sites may be detached from *mastergroup*. The new variables (see Fig. 6) are *groupid, partitionedgroup, connectionstatus, groupcounter, messagecounter, messageseqno*. CONNECTIONSTATUS is enumerated set containing the value ENABLE and DISABLE. The description of other variables are as follows:

- The variable *groupid* is used to assign unique id to a particular group. The variable *groupcounter* counts the id of a group. Each time when an id is assigned to a group its value is incremented by one. The variable *groupcounter* is initialized by 1 (see Fig. 7).
- The variable *partitionedgroup* contains the set of sites which are detached from main group named as *mastergroup*.
- The variable *messageseqno* is used to assign the sequence number of a message.
- The variable *connectionstatus* gives the status of connection between two groups of sites and show whether the connection is enable or disable.

4.1 Partitioning of sites (SITE_PARTITION event)

The specification of this event is given in Fig. 8. It models the partitioning of sites. During the detachment process a group of sites may be detached from the master group. The variable *partitionedgroup* contains those detached group. A unique group id is assigned to them. As a result it may form several disjoint groups having different group ids (*act2 and act3*). The group of sites which are detached before have smaller group id than those group which are detached after it. The connection status between the partitioned sites and master group is disabled by *act5*.

4.2 (PARTITION_CREATE_VIEW event)

The group of partitioned site may be in contact with different partitioned group. This event (see Fig. 9) sets the same group id if the partitioned group are in contact with each other (*act1)* and enable the connection status between those group (*act2*).

4.3 Delivery of messages after view creation (PARTITIONED_MESSAGE_DELIVER event)

The specification of this event is given in Fig. 10. After the creation of view the newly partitioned group may be in contact with those groups which are partitioned before it. So there is possibility that the sites in newly partitioned group may have some messages which are not in previously partitioned group (*see grd10 & grd11*). This event makes the delivery of those messages to the sites which have not these messages (*act1*).

4.4 Re-admission in master group (ADD_TO_MASTER event)

This event (see Fig. 11) models the re-admission of partitioned sites in to the master group. The group of sites from partitioned group may be re-join the master group. The connection status of this group and master group is enabled by *act1*.The *act4* makes the group id as same as master group.

After re-joining in the master group the total order delivery of messages is done by the sequencer present in master group through the ToDeliver event.

```
MACHINE
    partition_m
REFINES
    Total_m
SEES
    partition_c

VARIABLES
    sender, totalorder, tdeliver,
    mastergroup,groupcounter, groupid,
    partitionedgroup, messagecounter, messageseqno,
    connectionstatus

INVARIANTS
    inv1  :  groupcounter ∈ ℕ
    inv2  :  groupid ∈ ℙ(SITE)→ℕ
    inv3  :  partitionedgroup ∈ ℙ(mastergroup)
    inv4  :  messagecounter ∈ ℕ
```

```
inv5 :  messageseqno ∈ MESSAGE2ℕ
inv6 :  connectionstatus ∈ℙ(SITE)2 (ℙ(SITE) 2CONNECTIONSTATUS)
```

Fig 6: Machine variables and invariants of refinement model

```
INITIALISATION ≙
    BEGIN
        act1  :  sender ≔ ∅    act2: totalorder ≔ ∅
        act3  :  tdeliver ≔ ∅  act4:mastergroup ≔ ∅
        act5  :  groupcounter ≔ 1   act6: groupid ≔ ∅
        act7  :  partitionedgroup ≔ ∅
        act8  :  messageseqno ≔ ∅
        act9  :  messagecounter ≔0
        act10 :  connectionstatus≔∅
    END
```

Fig 7: Initialization of the machine variables

```
SITE_PARTITION ≙
    ANY
        ss
    WHERE
        grd1  :  ss∈ ℙ(mastergroup)
        grd2  :  ss ∉ ℙ(partitionedgroup)
    THEN
        act1  :  mastergroup≔ mastergroup∖ss
        act2  :  groupid(ss)≔groupcounter
        act3  :  groupcounter≔groupcounter+1
        act4  :  partitionedgroup≔
                  partitionedgroup ∪ ss
        act5:    connectionstatus(mastergroup)≔
                 connectionstatus(mastergroup)+
                 {ss↦ DISABLE}
    END
```

Fig 8: Specification of SITE_PARTITION event

```
PARTITION_CREATE_VIEW ≙
    ANY
        ss,tt
    WHERE
        grd1  :  ss∈ ℙ(partitionedgroup)
        grd2  :  tt∈ ℙ(partitionedgroup)      grd3:  ss≠tt
    THEN
        act1  :  groupid(ss)≔groupid(tt)
        act2  :  connectionstatus(ss) ≔
                 connectionstatus(ss)+{tt↦ ENABLE}
    END
```

Fig 9: Specification of PARTITION_CREATE_VIEW event

```
PARTITIONED_MESSAGE_DELIVER ≙
ANY
    ss, mm, tt
WHERE
        grd1  :  ss∈ SITE   grd2: tt∈ SITE   grd3:  ss≠tt
        grd4  :  ss∈ partitionedgroup
        grd5  :  tt∈ partitionedgroup
        grd6  :  groupid({ss})=groupid({tt})
        grd7  :  mm ∈ MESSAGE
        grd8  :  mm ∈ dom(sender)
        grd9  :  mm ∈ ran(tdeliver)
        grd10 :  ss↦mm ∈ tdeliver
        grd11 :  tt↦mm ∉ tdeliver
                 ∀m·(m∈ MESSAGE ∧
        grd12    (messageseqno(m)< messageseqno(mm))
          :      ⇒(tt↦m)∈ tdeliver)

    THEN
        act1  :  tdeliver≔ tdeliver ∪ {tt↦mm}
    END
```

Fig 10: PARTITION_MESSAGE_DELIVER event

```
ADD_TO_MASTER ≙
    ANY
        ss
    WHERE
        grd1  :  ss ∈ ℙ(partitionedgroup)
        grd2  :  ss∉ ℙ(mastergroup)
    THEN
        act1  :  connectionstatus(mastergroup)≔ connect
                 ionstatus(mastergroup)+{ss↦ ENABLE}
        act2  :  partitionedgroup≔ partitionedgroup∖ ss
        act3  :  mastergroup≔ mastergroup ∪ ss
        act4  :  groupid(ss)≔ groupid(mastergroup)
    END
```

Fig 11: Specification of ADD_TO_MASTER event

5. CONCLUSIONS

In this paper, we have presented a rigorous design of partition aware total order broadcast system using Event-B on Rodin platform. In the abstract model we have designed total order delivery of messages. There is a sequencer site which takes the responsibility of ordering of messages. In the refinement model, the specification of site partition is considered where a group of sites may be detached from the main group named as mastergroup. For each detached group a unique groupid is assigned to it. After some times when the partitioned group of sites re-joins the master group the total ordered delivery is done by sequencer present in master group. This work is carried out on Rodin platform which generates the proof obligations. These proofs are discharged automatically by the prover of the tool.

6. REFERENCES

[1] D´efago, X., Schiper, A., Urb´an, P. 2004 Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Comput. Surv. 36(4), 372–421.

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

[3] Suryavanshi, R., Yadav, D. 2010 Formal Development of Byzantine Immune Total Order Broadcast System using Event-B. In ICDEM 2010, F. Andres, R. Kannan, Eds. LNCS, Vol. 6411, Springer.

[4] Hinchey, M., Bowen, J.P. and Glass, R. 1996 Formal methods: Point-counterpoint. Computer, 29(4):18–19.

[5] Jones, C., Jackson, D. and Wing, J. 1996 Formal methods light. Computer, 29(4): 20–22.

[6] Saiedian, H. 1996 An invitation to formal methods, Computer, 29(4):16–17.

[7] Butler, M. 1997 An Approach to Design of Distributed Systems with B. In Proceedings 1997 10th Int. Conf. of Z Users: The Z Formal Specification Notation (ZUM), vol.1212, LNCS, pp.223-241.

[8] Butler, M., Walden, M. 1996 Distributed System Development in B. In Proceedings of Ist Conf. in B Method, Nantes, pp.155-168.

[9] Rezazadeh, A., Butler, M. 2005 Some Guidelines for formal development of web based application in B Method. In Proceedings of 4th Intl. Conf. of B and Z users, Guildford, LNCS, Springer, pp 472-491.

[10] Abrial, J.R. 1996 The B-Book: Assigning programs to meanings. Cambridge University Press.

[11] Abrial, J.R. 1996 Extending B without changing it for developing distributed systems. In First B Conference.

[12] Yadav, D., Butler, M. 2005 Application of Event B to Global Causal Ordering for Fault Tolerant Transactions. In Proceedings of REFT 2005, Newcastle upon Tyne, pp 93-103.

[13] Butler, M. and Yadav, D. An incremental development of the mondex system in Event-B. In Formal Aspects of Computing, 20(1):61–77, 2008.

[14] Abrial, J.R. and Voison, L. 2005 Event-B language. Technical Report, Deliverables 3.2, EU Project IST-511599-RODIN, http://rodin.cs.ncl.ac.uk/deliverables/D7.pdf.

[15] Abrial, J.R. 2007 A system development process with Event-B and the Rodin platform. In proceedings of ICFEM 2007, Butler, M., Hinchey, L. Petrie, Eds. vol.4789, LNCS, Springer, pp.1-3.

[16] Stanoi, I., Agrawal, D., Abbadi, A., 1998 Using broadcast primitives in replicated databases. In Proc. of 18th IEEE Intl. Conf. on Distributed Computing System,ICDCS, pages 148–155.

[17] Baldoni, R., Cimmino, S., Marchetti, C., 2005 Total order communications: A practical analysis. In EDCC 2005, Mario Dal Cin, Mohamed Kaˆaniche, and Andr´as Pataricza, Eds, LNCS Vol. 3463, pages 38–54. Springer.

[18] Birman, K., Schiper, A.,Stephenson, P., 1991 Lightweigt causal and atomic group multicast. ACM Trans. Comput. Syst., 9(3):272–314,.

[19] Marchetti, C., Cimmino, S., Baldoni, R., 2006 A classification of total order specifications and its application to fixed sequencer-based implementations. Journal of Parallel and Distributed Computing, 66(1):108–127.