Formal Verification of Causal Order-Based Load Distribution Mechanism Using Event-B

Pooja Yadav, Raghuraj Suryavanshi, Arun Kumar Singh and Divakar Yadav

1 Introduction

Distributed systems are very complex to understand and develop. There is a need to formally verify and ensure the correctness of distributed systems and algorithms. During last few years, the research in the field of formal methods has done significant work in the development of describing and analysing the complex systems in formal languages [1, 2]. In distributed systems, formal methods take an important role for ensuring the correctness of several protocols and algorithms. Formal verification is done either through model checking or theorem proving [3, 4]. Model checking is a model-oriented approach, which verifies the correctness of system automatically by traversing every possible execution path. It is expressed in terms of finite state automata that describe all possible transitions states. In this technique, for a given problem, a formal model describing its behavioural properties is developed in formal language. All the properties of models are verified. In order to verify all possible execution paths, it is required that the model must be finite. The problem may also appear when the model which is finite has considerable size. Theorem proving is the act of generating a mathematical proof for a mathematical statement to be true

P. Yadav
Abdul Kalam Technical University, Lucknow 226031, India
e-mail: poojayadav255@gmail.com

R. Suryavanshi (✉)
Pranveer Singh Institute of Technology, Kanpur 209305, India
e-mail: raghuraj_singh09@yahoo.co.in

A. K. Singh
Rajkiya Engineering College, Kannauj 209732, India
e-mail: aksingh_uptu@rediffmail.com

D. Yadav
Institute of Engineering and Technology, Lucknow 226021, India
e-mail: divakar_yadav@rediffmail.com

R. K. Shukla et al. (eds.), *Data, Engineering and Applications*,
https://doi.org/10.1007/978-981-13-6351-1_18

[3, 5, 6]. In this proving technique, system and its properties are specified in terms of mathematical logic. The verification of properties is done by discharging proof obligations generated by the system. If the proof is discharged for a statement, then it is known to be true and is said to be a theorem. The main advantage of theorem proving over model checking is that it can be used to verify the system having infinite states.

We have considered Event-B as a formal method for verification of our model. Event-B [7–9] is a formal technique, which is used to develop and formalize such system whose component can be modelled as discrete transition systems. It also provides refinement-based development of a complex model and has control systems within its scope. Event-B modelling can be used in various application areas like sequential programmes, concurrent programmes and distributed systems [10].

In this paper, we have developed formal model of distributed load migration mechanism using Event-B. Distributed system is a collection of autonomous systems connected by the network and they communicate with each other for the completion of common goal [11]. In this environment, the users submit the task at their sites for processing. The random arrival of tasks and their service order create the possibility that several sites may become heavily loaded and others may ideal or lightly loaded. It may degrade the performance of the whole system. Therefore, load distribution scheme is required for efficient use of resources and to enhance the performance [11–13]. In this paper, we have considered maximum load count value of site as the threshold value. This threshold value indicates maximum number of tasks that can be executed without affecting the performance of system. When a new task is submitted at site, the load count value of that site will be increased. When load count value of any site exceeds threshold value, then that site will become heavily loaded site. The load of this site should be transferred to idle or lightly loaded site. In order to find out low load site, heavily loaded site broadcasts load transfer request message to all sites. Site, whose load count value is lesser than threshold value, sends load reply message to sender. At any instance, lightly loaded site may receive number of load request messages from several heavily loaded sites. It will not send a reply message to all heavily loaded sites. It will send a reply only to that site first whose request for load transfer message arrived first. For ensuring ordered delivery of message, we have introduced a notion of causal order delivery [14]. The lightly loaded site will send a reply only to that site whose request for load transfer causally precedes the request from others. After receiving reply message from low load site, load from heavily loaded site will be transferred to it.

The remainder of this paper is organized as follows: Sect. 2 describes Event-B as formal method, Sect. 3 presents causal order broadcast, Sect. 4 outlines Event-B Model of causal order-based load distribution. This model consists of events like task submission event for submission of new task, enable and disable load transfer event for changing status of site when load count value increase and decrease from threshold value, broadcast and deliver event to formalize ordered delivery of load request message, reply message event, which models the sending of reply message to heavily loaded site and load reduction event to reduce load value. Finally, Sect. 5 concludes the paper.

2 Event-B

Event-B [14–18] is a formal technique, which captures complete system specifications on the basis of requirement and system behaviour. It can be expressed in form of states and events. It specifies the model in mathematical form. It defines mathematical structures as context and machine [19–21]. The context part of the model contains sets, constants and axioms, which are used to describe static properties of system. The dynamic part of the model is shown by machine part, which contains set of variables, invariants and events that modify the value of state variable when it triggers. The variables of model are constrained by invariants. The invariants of model which describe the properties of model should not be violated when an event occurs. The event contains guards which are necessary conditions for an event to trigger and list of actions. When all guards of an event become true, then set of actions written under it will be performed. The action of an event is expressed by substitution operation. It specifies how the state of system may change. The event may use local variables. The scope of that variable will be local to the event.

For ensuring correctness of the model, Event-B method requires to discharge all proof obligations generated by the model. Proof obligations serve to verify properties of the model.

There are several tools which support to write Event-B specifications. We have considered Rodin tool [22–24] for the development of our model. In order to discharge proof obligations generated by the model, there are various plug-ins which are provided by this tool. Event-B uses set-theoretic notations to specify the model. The syntax and detail description of Event-B notations can be found in [15].

3 Causal Order Broadcast

The formalization of causal relationship in distributed system was initiated by Lamport in [25]. Later, causal ordering of messages is proposed by Birman, Schiper and Stephenson [26]. The causal order property can be ensured by combining FIFO order and local order property [27].

FIFO order property says that if any site *Si* broadcasts a message *Ma* before broadcast another message *Mb*, then each receiving site delivers *Ma* before *Mb*.

Local order property says that if any site *Si* delivers message *Ma* before broadcasting message *Mb*, then every receiving site delivers *Ma* before *Mb*.

The causal order property says that if broadcasting of a message *Ma* causally precedes broadcasting of a message *Mb*, then delivery of message *Ma* at each site should be done before message *Mb* (Fig. 1).

We can say that message *Ma* causally precedes message *Mb* if send event of message *Ma*; *send(Ma*) at site *Si* happened before message sending event *send(Mb)* of message *Mb* at site *Sj*. The causality of the message can also be related with receive event of the message. A message *Ma* causally precedes *Mb* if receive event of

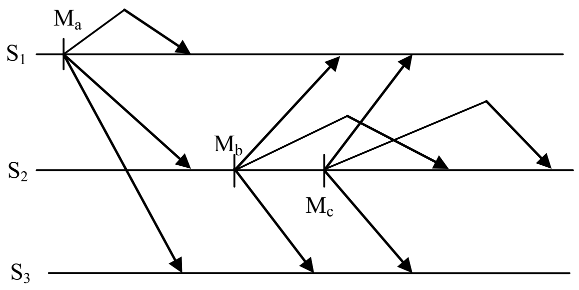

Fig. 1 Causal order broadcast

message *Ma* causally precedes the broadcast of *Mb*. As given in Fig. 1, broadcasting of message *Ma* causally precedes broadcasting of *Mb* and each recipient site delivers *Ma* before *Mb*. Similarly, broadcasting of message *Mb* causally precedes broadcasting of message *Mc* and each recipient site delivers *Mb* before *Mc*.

4 Event-B Model of Load Distribution Mechanism

We start with the distributed system model having a set of sites. Since there is no system-wide global clock or shared memory, the information from one site to other site is exchanged through messages. At any site, new task may be submitted. The task may be process or transaction which will perform some operation (reading or writing on data objects) at that site. Therefore, submission of new task will increase the load at that site. When the load count value exceeds a certain limit known as threshold value, then the performance of the system will degrade. In order to capture maximum throughput from the system, we need to distribute load from heavily loaded site to idle site in efficient manner.

In our model, the context part contains *SITE* and *MESSAGE* as carrier set. *TRANSFERSTATUS* and *TYPE* are declared as enumerated set. The set *TRANSFERSTATUS* represents load transfer status of site informs of disable and enable.

Initially, load transfer status of every site is disable because every site is underloaded. When load count value exceeds threshold value, then load transfer status will be set as enable. The set *TYPE* has element *LOAD_REQ* and *LOAD_REP* which is used to formalize type of message as load request and load reply, respectively. The machine part consists of variables, invariants and events. The description of variables is as follows (Fig. 2):

(i) The variable *load_at_site* is specified as total function. It represents load count value of every site.
(ii) The variable *loadtransferstatus* is a total function between site to *TRANSFERSTATUS*. Depending on the *loadcount* value, the load transfer status of every site may be either disable or enable.
(iii) The variable *threshold* value is declared as natural number.

Variables :

load_at_site, loadtransferstatus, threshold_value, sender, cdeliver, messagetype, messagesent, delorder, corder, replymsgsent

INVARIANTS

inv1: $load\_at\_site \in SITE \rightarrow \mathbb{N}$

inv2: $loadtransferstatus \in SITE \rightarrow TRANSFERSTATUS$

inv3: $threshold\_value \in \mathbb{N}$

inv4: $sender \in MESSAGE \nrightarrow SITE$

inv5: $cdeliver \in SITE \leftrightarrow MESSAGE$

inv6: $messagesent \subseteq MESSAGE$

inv7: $messagetype \in messagesent \rightarrow TYPE$

inv8: $corder \in MESSAGE \leftrightarrow MESSAGE$

inv9: $delorder \in SITE \leftrightarrow (MESSAGE \leftrightarrow MESSAGE)$

inv10: $replymsgsent \in (MESSAGE \leftrightarrow MESSAGE)$

inv11: $ran(cdeliver) \subseteq dom(sender)$

inv12: $dom(corder) \subseteq dom(sender)$

inv13: $ran(corder) \subseteq dom(sender)$

inv14: $\forall ss \cdot (ss \in SITE \land loadtransferstatus(ss)=enable \Rightarrow load\ at\ site(ss) > threshold\ value)$

Fig. 2 Variables and invariants of model

(iv) The variable *sender* is specified as partial function from *MESSAGE* to *SITE*. It models the sending of message *m* by site *s*.

(v) The variable *cdeliver* is declared as:

$$cdeliver \in SITE \leftrightarrow MESSAGE$$

The operator $\leftrightarrow$ defines the set of relations between *SITE* and *MESSAGE*. It represents causal delivery of messages at any site.

(vi) The variable *messagesent* represents the set of messages which have been sent.

(vii) The variable *messagetype* maps each sent message with its type. The type of message may be load request message or load reply message.

(viii) The variable *corder* models the causal relationship between messages. The mapping *(mmmm)* ∈ *corder* indicates that message *m* causally precedes message *mm*.

(ix) The delivery order of messages at any site is shown by *delorder*. It is declared as relation between site to set of ordered pair of messages. The mapping *ssm(mmmm)* ∈ *delorder* indicates that at site *ss* message, *m* is delivered before message *mm*.

(x) The variable *replymsgsent* models the sending of reply message corresponding to request message. The mapping *(m1mm2)* ∈ *replymsgsent* indicates that reply message *m1* has been sent corresponding to its request message *m2*.

4.1 Submission of Task

The event *TASK SUBMISSION* is given in Fig. 3. This event model the submission of task at any site *ss*. Every time when this event occurs, it increases the load count value of site by one. The action *act1* represents that load at site *ss* is incremented by one.

4.2 Enabling and Disabling Load Transfer Status

The event *ENABLE TRANSFER* updates load transfer status of site (Fig. 3). When load count value of any site exceeds threshold value, then this event updates load transfer status of that site as enable. The guard *grd2* ensures that load of site *ss* is greater than threshold value. The guard *grd3* ensures that load transfer status of site *ss* is disable. The action *act1* updates the load transfer status of site *ss* as enable.

The event *DISABLE TRANSFER* is given in Fig. 3. The guard *grd2* ensures that load count value of site *ss* is less than threshold value. The action *act1* set load transfer status of that site as disable.

4.3 Broadcasting and Delivery of Load Request Message

This event enables broadcasting of load request message to all sites (see Fig. 4). When the load count of any site exceeds its threshold value, then this site broadcast load request message to all sites. The purpose of the broadcasting request message is to know that which site is under loaded. The message that has not been sent is

```
TASK_SUBMISSION ≙
ANY ss
WHERE
grd1:  ss∈ SITE
THEN
act1:  load_at_site(ss):= load_at_site(ss)+1
END

ENABLE_TRANSFER ≙
ANY ss
WHERE
grd1:  ss∈ SITE
grd2:  load_at_site(ss)>threshold_value
grd3:  loadtransferstatus(ss)=disable
THEN
act1:  loadtransferstatus(ss):= enable
END
```

```
DISABLE_TRANSFER ≙
ANY ss
WHERE
grd1:  ss∈SITE
grd2:  load_at_site(ss)<threshold_value
THEN
act1:  loadtransferstatus(ss):= disable
END
```

Fig. 3 Task submission, enable transfer and disable transfer event

ensured by guard *grd2* and *grd4*. The load transfer status of site *ss* is enable and is ensured through guard *grd3*. Due to the occurrence of this event, message *mm* will be broadcast (*act2*). The message *mm* will be added to *messagesent* set (*act1*). The action *act3* set the status of message *mm* as load request message. In this event, we are also ensuring ordered delivery of request message at sending site. The action *act4* ensures that all those messages which are sent by site *ss* will causally precede message *mm* (FIFO Order). The action *act5* specifies delivery of message *mm* at site *ss*. The action *act6* gives delivery order of message.

Delivery of load request message is given in Fig. 5. The guard *grd3* and *grd4* ensure that message *mm* has been sent but it is not delivered to site *ss*. The guard *grd5* is written as

$$\forall m \cdot (m \in MESSAGE \wedge (m \mapsto mm) \in corder \Rightarrow (ss \mapsto m) \in cdeliver)$$

It ensures that all messages *m* which causally precede message *mm* have already been delivered at site *ss*. Delivery of message *mm* at site *ss* is ensured by action *act1*. The action *act2* makes the delivery order of messages at site *ss*.

```
BROADCAST ≙
ANY ss, mm
WHERE
grd1:  ss∈ SITE
grd2:  mm∉ messagesent
grd3:  loadtransferstatus(ss)=enable
grd4:  mm∉ dom (sender)
THEN
act1:  messagesent≔messagesent∪{mm}
act2:  sender≔sender∪ {mm↦ ss}
act3:  messagetype(mm)≔LOAD_REQ
act4:  corder≔corder ∪((sender∼[{ss}] × {mm})
       ∪(deliver[{ss}] × {mm}))
act5:  cdeliver≔ cdeliver∪ {ss↦ mm}
act6:  delorder ≔delorder ∪{ss↦(cdeliver[{ss}] × {mm})}
END
```

Fig. 4 Broadcast event

4.4 Sending of Reply Message

The event *REPLY* is given in Fig. 6. This event models sending of load reply message (*LOAD_REP*) by those sites whose load count value is lesser than threshold value. The load request message *mm* has been received by site *s* is ensured by guards *grd3 and grd4*. Site may receive number of load request messages from several heavily loaded sites but it will send a reply only to that site whose load request message *LOAD_REQ* message causally precedes other requests. The guard *grd5* ensures that load request message *mm* causally precedes all load request message *msg*. Therefore, site *s* will send load reply message *m* corresponding to load request message *mm* only. The load count value of site *s* is lesser than threshold value and is ensured by guard *grd6*. The reply message *m* has not been previously sent is ensured through guard *grd7 and grd8*. The action *act1* ensures sending of message *m* by site *s*. The action *act2* adds the message *m* to *messagesent* set. The action *act3* set the status

```
DELIVERY ≙
ANY ss,mm
WHERE
grd1:  ss∈ SITE
grd2:  mm∈ MESSAGE
grd3:  mm∈ dom(sender)
grd4:  (ss↦mm)∉cdeliver
grd5:    ∀m·(m ∈MESSAGE ∧ (m↦mm) ∈corder ⇒
         (ss ↦m) ∈cdeliver)
THEN
act1:  cdeliver≔ cdeliver∪{ss↦ mm}
act2:  delorder≔delorder ∪{ss↦(deliver[{ss}] × {mm})}
END
```

Fig. 5 Deliver event

of message *m* as *load reply (LOAD_REP).* The action *act4* makes the entry of reply message *m* corresponding to request message *mm.*

4.5 Load Reduction Event

This event model the load reduction from heavily loaded site (Fig. 7). The guard *grd2* ensures that site *ss* is heavily loaded because its load count value is greater than threshold value. Guards *grd3, grd4 and grd5* ensure that site *ss* has received the load reply message *m.* Receiving of reply message also indicates that there is some site whose load count value is less than threshold value. Request message *mm* sent by site *ss* is ensured through guards *grd6, grd7 and grd8*. The guard *grd9* ensures that message *m* is reply message of request message *mm*. Due to the occurrence of this event, load count value of site *ss* is reduced by one (*act1*). After the reduction of load, it will be submitted at low load site in the form of task through TASK SUBMISSION event.

```
REPLY $\triangleq$
ANY s, mm, m
WHERE
grd1:  $s \in SITE$
grd2:  $mm \in messagesent$
grd3:  $messagetype(mm)=LOAD\_REQ$
grd4:  $s \mapsto mm \in cdeliver$
grd5:  $\forall msg \cdot (msg \in messagesent \land messagetype(msg)$
       $=LOAD\_REQ \Rightarrow (mm \mapsto msg) \in corder)$
grd6:  $load\_at\_site(s) < threshold\_value$
grd7:  $m \in MESSAGE$
grd8:  $m \notin dom(sender)$
THEN
act1:  $sender := sender \cup \{m \mapsto s\}$
act2:  $messagesent := messagesent \cup \{m\}$
act3:  $messagetype(m) := LOAD\_REP$
act4:  $replymsgsent := replymsgsent \cup \{m \mapsto mm\}$
END
```

Fig. 6 Reply event

5 Conclusion

Distributed systems provide tremendous processing capacity. In order to maximize the performance of the system, good load transfer or task migration schemes are required. The random arrival order of task and its random system service may create the situation that all resources and systems may not properly be utilized. Due to uneven load distribution, few of the sites may become heavily loaded and others may be ideal. We have introduced causal order delivery of load transfer request message which ensures ordered service of load request message.

In this paper, a formal development of causal order-based load distribution mechanism is done. Formal methods are mathematical techniques to verify the correctness of system properties. We have considered Event-B as a formal method for the devel-

Fig. 7 Load reduction event

```
LOAD_REDUCTION ≙
ANY ss, m, mm
WHERE
grd1:  ss∈ SITE
grd2:  load_at_site(ss)>threshold_value
grd3:  m∈ messagesent
grd4:  messagetype(m)=LOAD_REP
grd5:  ss↦m∈ cdeliver
grd6:  mm∈ messagesent
grd7:  mm↦ss∈ sender
grd8:  messagetype(mm)=LOAD_REQ
grd9:  (m↦mm)∈ replymsgsent
THEN
act1:  load_at_site(ss):= load_at_site(ss)−1
END
```

opment of our model. In Event-B model, the properties of model are specified through invariants. These invariants should not be violated during the execution of the model. Event-B model generates proof obligations, and we need to discharge all proofs generated by it. A total of 42 proof obligations are generated by the model out of which 30 proofs are discharged automatically whilst 12 proofs are discharged interactively. In order to ensure correctness, we have added the following invariants:

$$\text{ran(cdeliver) (dom(sender)} \ldots \text{(inv11)}$$

$$\text{dom(corder) (dom(sender)} \ldots \text{(inv12)}$$

$$\text{ran(corder) (dom(sender)} \ldots \text{(inv13)}$$

$$\text{! ss(ss} \in \text{SITE \& loadtransferstatus(ss)} = \text{enable)G load at site(ss)} > \text{(threshold value)} \ldots \text{(inv1}$$

The invariant inv11 ensures that messages which are delivered should be a subset of messages which have been sent. Similarly, invariants inv12 and inv13 ensure that messages for which ordering is maintained (causal order) should be a subset of sent messages. The invariant inv14 ensures that if load transfer status of any site is enable, then load of that site exceeds threshold value of that site.

The invariants and proofs of model give a clear insight of model. In future, we plan to strengthen invariants and add fault-tolerance property to this model.

References

1. Bjrner, D.: Logics of formal specification languages. Comput. Inform. **22**(1–2), This double issue contains the following papers on B, CafeOBJ, CASL, RAISE, TLA+ and Z (2003)
2. Bjrner, D.: Special double issue on formal methods of program development. Int. J. Softw. Inform. **3** (2009)
3. Shankar, N.: Combining theorem proving and model checking through symbolic analysis. In: Proceeding of CONCUR ’00, vol. 1877, pp. 1–16. LNCS, Springer (2000)
4. Fitzgerald, J., Larsen, P.G.: Modelling Systems—Practical Tools and Techniques in Software Development. Cambridge University Press, Cambridge, UK, Second edition (2009)
5. Clarke, E., Zhao, X.: A theorem prover for mathematica. In automated deduction-CADE-II. In: 11th International Conference on Automated Deduction, pp. 761–763. Saratoga Springs, New York, 15–18 June 1992
6. Clarke, E., Zhao, X.: A theorem prover for Mathematica. Math. J. (1993)
7. Abrial, J., Butler,M., Hallerstede,S., Voisin, L.: An open extensible tool environment for Event-B. In: Liu, Z., He, J. (eds.) ICFEM, Lecture Notes in Computer Science, vol. 4260, pp. 588–605. Springer (2006)
8. Abrial, J.R.: Modeling in Event-B: System and Software Engineering. CambridgeUniversity Press (2010)
9. Abrial, J.R., Hallerstede, S.: Refinement, decomposition, and instantiation of discrete models. Appl. Event B Fundam. Inform. **77**(1–2), 1–28 (2007)
10. Butler, M.: An approach to the design of distributed systems with B AMN. In: Bowen, J.P., Hinchey, M.G., Till, D. (eds.) ZUM, Lecture Notes in Computer Science, vol. 1212, pp. 223–241. Springer (1997)
11. Singhal, M., Shivratri, N.G.: Advanced Concepts in Operating Systems. Tata McGraw-Hill Book Company (2012)
12. Lazowska, D.E., Zahorjan, J.: Adaptive load sharing in homogeneous distributed systems. IEEE Trans. Softw. Eng. **12**(5), 662–675 (1986)
13. Lazowska, D.E., Zahorjan, J.: A Comparison of receiver-initiated and sender-initiated adaptive load sharing. Perform. Eval. **6**(1) 53–68 (1986)
14. Yadav, D., Butler, M.: Application of Event B to global causal ordering for fault tolerant transactions. In: Proceeding of Workshop on Rigorous Engineering of Fault Tolerant System, REFT05, Newcastle upon Tyne, pp. 93–103, 19 July 2005
15. Yadav, D., Butler, M.: Rigorous design of fault-tolerant transactions for replicated database systems using Event B. In: Butler, M., Jones, C.B., Romanovsky, A, Troubitsyna, E. (eds.) Rigorous Development of Complex Fault-Tolerant Systems. Lecture Notes in Computer Science, vol. 4157, pp. 343–363. Springer, Heidelberg (2006)
16. Yeganefard, S., Butler, M., Rezazadeh, A.: Evaluation of a guideline by formal modelling of cruise control system in Event-B. Proc. NFM **2010**, 182–191 (2010)
17. Liu, J., Liu, J.: A formal framework for hybrid Event B. Electron. Notes Theor. Sci. **309**(2014), 3–12 (2014) (Elsevier)
18. Suryavanshi, R., Yadav, D.: Formal development of byzantine immune total order broadcast system using Event-B. In: Andres, F., Kannan, R. (eds.) ICDEM 2010. LNCS, vol. 6411, pp. 317–324. Springer, Germany (2010)
19. Hallerstede, S., Leuschel, M.: Experiments in program verification using Event-B. Form. Asp. Comput. **24**, 97–125 (2012)

20. Suryavanshi, R., Yadav, D.: Rigorous design of lazy replication system using Event-B. In: Communications in Computer and Information Science, vol. 0306, pp. 400–411. Springer, Germany (2012). ISSN 1865-0929
21. Suryavanshi, R., Yadav, D.: Modeling of multiversion concurrency control system using Event-B. In: Federated Conference on Computer Science and Information systems (FedCSIS), Poland, indexed and published by IEEE, pp. 1397–1401, 9–12 Sept 2012. ISBN 978-83-60810-51-4
22. Banach, R.: Retrenchment for Event-B: usecase-wise development and Rodin integration. Form. Asp. Comput. **23**, 113–131 (2011)
23. Abrial, J.R., Cansell, D., Mery, D.: A mechanically proved and incremental development of ieee 1394 tree identify protocol. Form. Asp. Comput. **14**(3), 215–227 (2003)
24. Metayer, C., Abrial,J.R., Voison, L.: Event-B language. RODIN deliverables 3.2. http://rodin.cs.ncl.ac.uk/deliverables/D7.pdf (2005)
25. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **25**(7), 558–565 (1978)
26. Birman, K., Schiper, A., Stephenson, P.: Lightweight causal and atomic group multicast. ACM Trans. Comput. Syst. **9**(3), 272–314 (1991)
27. Yadav, D., Butler, M.: 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 (2007)