Formal Development of Broadcast Systems and Verification of Ordering Properties using Event-B

Divakar Yadav and Michael Butler

Divakar Yadav is Assistant Professor of Computer Science at Institute of Engineering and Technology, Lucknow, India. This research was carried out at Dependable Systems and Software Engineering Research group, School of Electronics and Computer Science, University of Southampton as a Commonwealth Scholar supported by the Commonwealth Scholarship Commission in the United Kingdom.

Michael Butler is Professor of Computer Science at Dependable Systems and Software Engineering research group, School of Electronics and Computer Science, University of Southampton, UK. Michael Butler’s contribution is part of the EU ICT project 214158 DEPLOY (Industrial deployment of system engineering methods providing high dependability and productivity) www.deploy-project.eu.

September 18, 2008
Abstract

The use of formal methods to develop a model of a system, specifying critical properties and the verification of them is a way of obtaining better design of dependable services. Event-B is a formal technique for the development of models of distributed systems. This technique consists of describing rigorously the problem in an abstract model, introducing solutions or the design details in refinement steps to obtain more concrete specifications, and verifying that the proposed solutions are correct. This technique requires the discharge of proof obligations for consistency checking and refinement checking. The existing B tools provide a significant automated proof support for generation of the proof obligations and discharging them. In this paper, we present a refinement approach to the development of models of distributed systems using Event-B where processes communicate by broadcasting and messages are delivered in a causal order and total order. In order to discharge proof obligations we discover some interesting invariants that describes the relationship of abstract causal and total order with the vector clock rules. Due to powerful tool support for Event-B we also get an impressive degree of automatic proof.

Index Terms


I. INTRODUCTION

Distributed systems are difficult to understand, build and reason about due to unavoidable concurrency [1]. In a fully asynchronous message passing system, there is no natural ordering of the messages. In such systems there is no concept of real time and it is assumed that messages are eventually delivered and processes eventually respond, but no assumption on time can be made. Group communication primitives provide ordering guarantees on the delivery of messages to different processes. Group communication primitives have been used as a basic building block for the development of reliable fault tolerant distributed applications [2]. Solutions based on group communication are used in the real world. For example, ISIS [3] based solutions are used at the New York Stock Exchange for providing reliable multicast communication, at the Swiss Electronic Bourse and for developing a new generation of the French Air Traffic Control System [4]. These primitives have also been proposed for processing transactions and managing replicated databases [5], [6], [7]. The total order [2] broadcast is one primitive which ensures that a message is delivered to the different recipient processes in the same order. Total order alone does not guarantee that messages are delivered in the order they were sent. The causal order broadcast [2], [8] provide guarantees that delivery order is also consistent with the order they were sent. In this paper we present
a formal development of a broadcast system that guarantees a total order conforming with the causal dependencies. The work presented in the paper constitutes a part of our work on formal development of transactions for replicated databases [9].

Our approach of gradual development of the broadcast system is based on the technique of abstraction and refinement. The important feature of this approach is to formally define an abstract global model of a system independent of the architecture and successively refine it to a detailed distributed design in a series of intermediate steps. This technique is supported by Event-B [10], [11], which is a variant of the B Method [12]. This formal technique consists of the following steps:

- Rigorous abstract description of the problem.
- Introduce solutions or design details in refinement steps to obtain more concrete specifications.
- Verifying that proposed refinements are valid.

This formal approach supports a step-wise development from initial abstract specifications to a detailed design of a system in the refinement steps. Through refinement we verify that the design of a system conforms to the abstract specifications. The B tools provide a significant automated proof support for generating the proof obligations and discharging them. This technique requires the discharge of proof obligations for consistency checking and refinement checking. The technique is supported by several B tools such as Rodin [13], B4Free [14], Atelier-B [15], B-ToolKit [16] and Click’n’Prove [17] that provide a significant automated proof support for generation of the proof obligations, factorizing complex proof obligations into simpler proofs and discharging them. The majority of the proof obligations are proved by the automatic prover of the tools. However, some complex proof obligations require user guidance through the interactive prover. These proof obligations also help in discovery of new system invariants. The proof obligations and the invariants help to understand the complexity of the problem and the correctness of the solutions. They also provide a clear insight into the system and enhance our understanding of why a design decision should work. We have used the Click’n’Prove [17] B tool for proof obligation generation and to discharge them. The essential features of the modelling and proof guidelines to obtain an high degree of automated proof for an Event-B development are outlined in [18]. Due to powerful support of B tools in discharging proof and our modelling style, we have achieved high degree of automatic proofs in this development. The proof statistics for this development is given later in this paper.

In this paper, we begin with incremental development of a system of causal order broadcast. In the abstract model of this system, we outline how an abstract causal order is constructed on the messages. The abstract causal order is defined by combining the properties of both FIFO and local order [19].
the refinement steps we outline how an abstract causal order can correctly be implemented by a system of vector clocks [20], [21], [22], [23]. Our approach for building causal order on the messages is based on the protocol reported in ISIS [8] that realizes causal order using vector clocks. We also verify that these models of the system preserve required ordering properties. A series of invariants is discovered while discharging the proofs due to refinement checking that describes the relationship of abstract causal order and the vector clocks rules. These invariants help explain why a system of vector clocks correctly implements causal order. Further, in a separate development, we extend the abstract model of a causal order broadcast to a system of total causal order broadcast such that the delivery of the messages also satisfies a total order on the messages in addition to a causal order. A total causal order broadcast not only preserves the causality among the messages but also delivers them in a total order. This model is based on the Broadcast Broadcast variant of a fixed sequencer algorithm [2] and uses a notion of the sequencer that constructs an abstract total order on the messages. In the refinement, the causal order is implemented by using a vector clock while the total order is implemented using the sequence numbers.

The remainder of the paper is organized as follows: Section II presents our modelling approach and informally presents various ordering properties, Section III presents incremental development of a system of causal order broadcast, Section IV present development of a system of total causal order broadcast and Section V compare our approach to other related work. Finally Section VI concludes the paper.

---

Fig. 1. Message Ordering

---

\[1\] A reliable broadcast that satisfies both causal and total order is also known as causal atomic broadcast.
II. PRELIMINARIES

In this section we outline the informal specifications of the message ordering properties and introduces the Event-B system.

A. Ordering Properties

The concept of a reliable broadcast [2], [19] is central to ordered broadcasts. A reliable broadcast is defined in terms of two primitives called broadcast and deliver, and it imposes no restriction on the order in which messages are delivered to the processes. A reliable broadcast can be used to deliver messages to processes following a fifo order, local order, causal order or total order providing ordering guarantees on the message delivery. Informal specifications of these ordering properties are given below.

1. FIFO Order: If a particular process broadcasts a message $M_1$ before it broadcasts a message $M_2$, then each recipient process delivers $M_1$ before $M_2$.

2. Local Order: If a process delivers $M_1$ before broadcasting the message $M_2$, then each recipient process delivers $M_1$ before $M_2$.

3. Causal Order: If the broadcast of a message $M_1$ causally precedes the broadcast of a message $M_2$, then no process delivers $M_2$ unless it has previously delivered $M_1$. If $M$ is broadcast by $P$, then all other messages sent by or delivered at $P$ will causally precedes $M$.

4. Total Order: If two processes $P_1$ and $P_2$ both deliver the messages $M_1$ and $M_2$ then $P_1$ delivers $M_1$ before $M_2$ if and only if $P_2$ delivers $M_1$ before $M_2$.

The FIFO and local order are outlined in Fig. 1(a) and 1(b) respectively. The dotted lines shows the violations of the respective ordering properties. A causal order broadcast delivers the messages respecting their causal precedence. However, if the broadcast of any two messages is not related by causal precedence (parallel messages), then it does not impose any requirement on the order in which they can be delivered. As shown in the Fig. 1(c), the broadcast of messages $M_1$ and $M_2$ are not related by a causal precedence relationship and the causal order broadcast delivers them to the processes in arbitrary order.

A total order broadcast is a reliable broadcast that satisfies the total order requirement. The agreement property of a reliable broadcast and the total order requirements of a total order broadcast requires that

---

2The causal order is defined by combining the properties of both FIFO and local order [19] and it is based on the notion of the happened before relationship [24], [25].

3The total order broadcast is also known as atomic broadcast.
all correct processes eventually deliver the same sequence of messages [19]. Since a total order defines an arbitrary ordering on the delivery of messages, it does not satisfy causal relations. A total causal order broadcast is a reliable broadcast that satisfies both causal and total order. Consider following two cases given in Fig. 1(d) and 1(e) that illustrate the causal order and the total order relationship. As shown in Fig. 1(d), all messages are delivered conforming to both the causal and the total order. However, as shown in Fig. 1(e), a broadcast satisfies a total order on the messages but does not preserve the causal relationships among them.

B. Event-B

The notion of abstract machine and refinement is central to Event-B. An abstract machine consists of sets, constants and variable clauses modelled as set theoretic constructs. The invariants and properties are defined as first order predicates. The event system is defined by its state and contains a number of events. The state of the system is defined by the variables and the events consists of guarded actions defined on the variables. The invariants state the properties that must be satisfied by the variables and maintained by the activation of the events. In refinement steps, guards may be strengthened, new events may be introduced and variables may added or removed. Abstract and concrete variables are related through gluing invariants. This technique requires the discharge of the proof obligations for consistency checking and refinement checking. Consistency checking involves showing that a machine preserves the invariants when events are invoked. Refinement checking involves showing that the specifications at each refinement step are valid.

Event-B notation is based on set theory and most of it is self explanatory. Some of the frequently used notations in our models are explained here to enhanced the readability. Let $A$ and $B$ be two sets, then the relational constructor ($\leftrightarrow$) defines the set of relations between $A$ and $B$ as :

$$A \leftrightarrow B = \mathcal{P}(A \times B)$$

where $\times$ is cartesian product of $A$ and $B$. A mapping of element $a \in A$ and $b \in B$ in a relation $R \in A \leftrightarrow B$ is written as $a \mapsto b$.

The domain of a relation $R \in A \leftrightarrow B$ is the set of elements of $A$ that $R$ relates to some elements in $B$ defined as :

$$\text{dom}(R) = \{a \mid a \in A \land \exists b. (b \in B \land a \mapsto b \in R)\}$$

Similarly, the range of relation $R \in A \leftrightarrow B$ is defined as set of elements in $B$ related to some element
in A defined as:
\[
\text{ran}(R) = \{b \mid b \in B \land \exists a. (a \in A \land a \mapsto b \in R)\}
\]

A relation \( R \in A \leftrightarrow B \) can be projected on a domain \( U \subseteq A \) called domain restriction \((\preceq)\) defined as
\[
U \preceq R = \{a \mapsto b \mid a \mapsto b \in R \land a \in U\}
\]

The domain anti-restriction \((U \preceq R)\) is defined as:
\[
U \preceq R = \{a \mapsto b \mid a \mapsto b \in R \land a \notin U\}
\]

The Relational image \( R[U] \) where \( U \subseteq A \) is defined as:
\[
R[U] = \{b \mid \exists a \cdot a \mapsto b \in R \land a \in U\}
\]

The relational inverse \((R^{-1})\) of a relation \( R \) is defined as:
\[
R^{-1} = \{b \mapsto a \mid a \mapsto b \in R\}
\]

If \( R_0 \in A \leftrightarrow B \) and \( R_1 \in A \leftrightarrow B \) are relations defined on set \( A \) and \( B \), the relational over-ride operator \((R_0 \preceq R_1)\) replaces mappings in relation \( R_0 \) by those in relation \( R_1 \).
\[
R_0 \preceq R_1 = (\text{dom}(R_1) \preceq R_0) \cup R_1
\]

A function is a relation with certain restrictions. The function may be a partial function \((\mapsto)\) or a total function \((\rightarrow)\). A partial function from set \( A \) to \( B \) \((A \mapsto B)\) is a relation which relates an element in \( A \) to at most one element in \( B \).

A total function from set \( A \) to \( B \) \((A \rightarrow B)\) is a partial function where \( \text{dom}(f) = A \), i.e., each element of the source set \( A \) is related to exactly one element of the target set. Given \( f \in A \rightarrow B \) and \( a \in \text{dom}(f) \), \( f(a) \) represents the unique value that \( a \) is mapped to by \( f \).

An injective function never maps two different elements of the source set to the same element of the target set. A partial injection from set \( A \) to \( B \) \((A \mapsto\rightarrow B)\) may be defined as,
\[
A \mapsto\rightarrow B = \{f \mid f \in A \rightarrow B \land f^{-1} \in B \rightarrow A\}
\]

### III. Causal Order Broadcast

In this section we present an incremental development of a system of causal order broadcast consisting of four levels of refinement chain.
A. Outline of the refinement steps

In this development we begin with an abstract model of a reliable broadcast and successively refine it to a model with vector clocks. A brief outline of each level is given below.

L0 This consists of an abstract model of a reliable broadcast. In this model processes communicate by broadcast and messages are delivered to each process only once including the sender.

L1 In this refinement, we outline how an abstract causal order is constructed by the sender. An abstract causal order is constructed by combining FIFO and local ordering properties.

L2 In this refinement, we introduce the notion of vector clocks. The abstract causal order is replaced by the vector clocks rules. We also discover gluing invariants which define the relationship of abstract causal order and vector rules.

L3 In this refinement, we present a simplification of the vector rules for updating the vector clock of recipient processes.

B. Abstract model of a reliable broadcast: Level-0

The abstract model of a reliable broadcast system is given as a B machine in the Fig. 2. The PROCESS and MESSAGE sets define types for the model. The variable sender is defined as a partial function from MESSAGE to PROCESS in invariant I-1. It contains mappings from MESSAGE to PROCESS. The mapping \((m \mapsto p) \in \text{sender}\) indicates that message \(m\) was sent by process \(p\). The partial function ensures that a message is sent by only one process. The variable cdeliver is a relation between PROCESS and MESSAGE defined in invariant I-2. A mapping of form \((p \mapsto m) \in \text{cdeliver}\) indicates that a process \(p\) has delivered a message \(m\). The sender and cdeliver are initialized as empty set.

In this machine, a broadcast message is delivered to its sender as well as all other processes. It may be noticed that all delivered messages must be messages that have been sent. This property is defined as invariant I-3. The events of sending and delivery of messages are shown as parameterized operations Broadcast(pp,mm) and Deliver(pp,mm). Note that the messages are not yet ordered in the abstract model. Also, a message \(mm\) is delivered to the sender at the time of broadcast. When a Broadcast event is invoked, the pair \((mm \mapsto pp)\) is added to function sender. The Deliver event is guarded by predicates. These predicates ensure that a process only delivers a message that has already been sent and that message has not been delivered to that process already.
C. Introducing global causal ordering on messages : Level-1

The refinement of the abstract model of broadcast is given in Fig. 3. As shown in the figure, the abstract causal order is represented by a variable \( corder \). A mapping of the form \( (m1 \mapsto m2) \in corder \) indicates that message \( m1 \) causally precedes \( m2 \). The invariant corresponding to variable \( corder \) is given as I-4. In order to represent the delivery order of messages at a process, variable \( delorder \) is used. A mapping \( (m1 \mapsto m2) \in delorder(p) \) indicates that process \( p \) has delivered \( m1 \) before \( m2 \). The typing invariant for variable \( delorder \) is given as I-5. Causal order can be defined only on those messages that have been sent. The corresponding invariants are given as I-6 and I-7.

The events \( Broadcast(pp,mm) \) and \( Deliver(pp,mm) \) respectively model the events of broadcasting a message and the causally ordered delivery of a message. As shown in the definition of the \( Broadcast \) event, a causal order is built by combining a FIFO order and a local order. When a process \( pp \) broadcasts a message \( mm \), the variable \( corder \) is updated by the mappings in \( (sender^{-1}\{pp\} \times \{mm\}) \). This indicates that all messages sent by \( pp \) before broadcasting \( mm \) causally precede \( mm \) conforming to the FIFO order. Similarly, the mappings in \( (cdeliver\{pp\} \times \{mm\}) \) indicate that the messages causally delivered to the process \( pp \) before broadcasting \( mm \) also causally precedes \( mm \) conforming to a local order.
On the occurrence of the Broadcast event, variable sender is updated with corresponding entries of the sender process and the message. The guard $mm \notin \text{dom}(\text{sender})$ ensures that each time a fresh message is broadcast. In the Deliver event, a process $pp$ delivers a message $mm$ only when all messages which causally precedes $mm$ are delivered. The guards of this event also ensure that a message is delivered only once.

1) Invariant properties of causal order: After developing the abstract model of causal order, our goal was to verify that this model also preserves the following critical properties. (1) The causal order properties informally defined in the Section II-A are preserved. (2) The causal order is transitive. (3) The causal order is non-symmetric and non-reflexive. In order to construct an invariant for the causal ordering property, we considered following property $^4$ relating abstract causal order and delivery order :

$$m_1 \leftrightarrow m_2 \in \text{delorder}(p) \Rightarrow m_1 \leftrightarrow m_2 \in \text{corder}$$  \hspace{1cm} (1)

$^4$We have omitted the quantifications over all identifiers ($m_1, m_2, p$ etc) to avoid clutter.
Before proving this property using the theorem prover we used the ProB [26] model checker and animator to precisely understand whether it is an invariant property. The ProB tool supports automatic consistency checking of B machines via model checking. However, the for exhaustive model checking the given sets of the machine must be restricted to small finite sets and integer variables must be limited to small numeric ranges. Major challenge for ProB is the problem of state space explosion, where linear increase in the size of specifications leads to exponential increase in the number of states, thereby, checking the larger B specifications becomes intractable. The main use of the ProB as a complement to interactive proof is that errors that results in counterexamples must be eliminated before interactive proof efforts. The interactive proofs also gives more insight into the design decision and discovered gluing invariants help explain why a design decision does work.

The property at (1) state that if messages $m_1$ and $m_2$ are delivered at process $p$ such that $m_1$ is delivered before $m_2$, then $m_1$ precedes $m_2$ in abstract causal order. The ProB suggested that (1) is not an invariant property because parallel messages delivered to different processes may have same delivery order. After experimentation with the (Pro B), we arrive at the following property :

$$m_1 \rightarrow m_2 \in corder \land p \rightarrow m_2 \in cdeliver$$

$$\Rightarrow m_1 \rightarrow m_2 \in delorder(p)$$

(2)

Property (2) states that if two messages are causally ordered then their delivery order will be same as their causal order only if that process has delivered the later message. Another property which we want to prove is that the causal order is transitive. To verify this we add following invariant :

$$m_1 \rightarrow m_2 \in corder \land m_2 \rightarrow m_3 \in corder$$

$$\Rightarrow m_1 \rightarrow m_3 \in corder$$

(3)

Lastly, to verify that the abstract causal order is non-symmetric and non-reflexive, we add following invariant :

$$m_1 \rightarrow m_2 \in corder \Rightarrow m_2 \rightarrow m_1 \notin corder$$

(4)

$$m \in MESSAGE \Rightarrow m \rightarrow m \notin corder$$

(5)

In the next step we add the properties given at (2), (3), (4) and (5) to this refinement as primary invariants and verify that these invariants are preserved by the model.
2) Proof obligations and invariant discovery: In this section we outline how we verify that the refined model \textit{CausalOrder} preserves the invariant properties given at (2), (3), (4) and (5). We also outline how the proof obligations generated by the B tool and the interactive prover guide us constructing new invariants. The primary invariant properties of the model of causal order broadcast system are given in Fig. 4 as \(I-8\), \(I-9\), \(I-10\) and \(I-11\). To prove that \(I-8\) is preserved by this refined model, we add this invariant to our model. After addition of this invariant to the model, the B tool generates two proof obligations associated with events \textit{Broadcast} and \textit{Deliver}. These proof obligations are discharged by the interactive prover without having to add new invariants. In the next step, we add invariant \(I-9\) to our model. When this invariant is added to the model, the B tool generates the following complex proof obligation associated with the \textit{Broadcast} event.

\[
\text{Broadcast}(pp, mm)PO1
\]

\[
\begin{align*}
I-9 & \land \\
mm & \notin \text{dom}(sender) \\
m1 \mapsto m2 & \in (\text{corder} \cup (\text{sender}^{-1}[\{pp\}] \times \{mm\})) \\
& \cup (\text{cdeliver}[\{pp\}] \times \{mm\})) \\
m2 \mapsto m3 & \in (\text{corder} \cup (\text{sender}^{-1}[\{pp\}] \times \{mm\})) \\
& \cup (\text{cdeliver}[\{pp\}] \times \{mm\})) \\
\Rightarrow \\
m1 \mapsto m3 & \in (\text{corder} \cup (\text{sender}^{-1}[\{pp\}] \times \{mm\})) \\
& \cup (\text{cdeliver}[\{pp\}] \times \{mm\}))
\end{align*}
\]
The proof obligation $PO1$ is reduced to the following simplified proof obligation using the interactive prover.

\[\text{Broadcast}(pp, mm) \quad PO2\]
\[
\begin{align*}
& m1 \rightarrow m2 \in \text{corder} \\
& m2 \in (\text{sender}^{-1}\{\{pp\}\}) \\
& m1 \notin (\text{sender}^{-1}\{\{pp\}\}) \\
& \Rightarrow m1 \in (\text{cdeliver}\{\{pp\}\})
\end{align*}
\]

The proof obligation $PO2$ states that if a message $m1$ causally precedes $m2$, i.e., $(m1 \rightarrow m2) \in \text{corder}$, and $pp$ is the sender of $m2$ and $m1$ was not sent by process $pp$ then process $pp$ must have delivered $m1$. After analyzing the proof obligation $PO2$, we construct following invariant and add it to our model.

\[
\begin{align*}
& m1 \rightarrow m2 \in \text{corder} \\
& m2 \in (\text{sender}^{-1}\{\{p\}\}) \\
& \Rightarrow m1 \in (\text{sender}^{-1}\{\{p\}\}) \lor m1 \in (\text{cdeliver}\{\{p\}\})
\end{align*}
\]

This invariant state that if $m1$ causally precedes $m2$ and $p$ is the sender of $m2$ then either $p$ is sender of $m1$ or $p$ has delivered $m1$. This invariant is given as $I-12$ in the Fig. 5. After adding invariant $I-12$ to the model we discharge the proof obligation $PO2$ associated with the Broadcast event. However, due to the addition of $I-12$, additional proof obligations associated with the Broadcast and Deliver events are generated. The proof obligation associated with the Broadcast event is discharged using the interactive prover. The following proof obligation associated with Deliver event cannot be discharged interactively.

\[\text{Deliver}(pp, mm) \quad PO3\]
\[
\begin{align*}
& I-12 \land \\
& m1 \rightarrow m2 \in \text{corder} \\
& m2 \in (\text{cdeliver}\{\{pp\}\}) \\
& \Rightarrow m1 \in (\text{cdeliver}\{\{pp\}\})
\end{align*}
\]

The $PO3$ states that for messages $m1$ and $m2$ where $m1$ causally precedes $m2$ and a process $pp$ has delivered $m2$ then $pp$ has also delivered $m1$. To discharge the proof obligation $PO3$ we construct following invariant and add it to our model.

\[
\begin{align*}
& m1 \rightarrow m2 \in \text{corder} \\
& p \rightarrow m2 \in \text{cdeliver} \\
& \Rightarrow p \rightarrow m1 \in \text{cdeliver}
\end{align*}
\]

This invariant state that if $m1$ causally precedes $m2$ then a process $p$ that has delivered $m2$, has also delivered $m1$. This invariant is given as $I-13$ in the Fig. 5. We notice that after adding invariant $I-13$
to the model we are able to discharge $PO3$. Addition of $I-13$ to the model also generates new proof obligations associated with $Broadcast$ and $Deliver$ events. These proof obligations are also discharged interactively using interactive prover.

<table>
<thead>
<tr>
<th>Invariants</th>
<th>Required By</th>
</tr>
</thead>
<tbody>
<tr>
<td>/<em>I-12</em>/ $(m1 \rightarrow m2) \in corder \land m2 \in sender^{-1}[[p]]$</td>
<td>$Broadcast$, $Deliver$</td>
</tr>
<tr>
<td>$\Rightarrow (m1 \in sender^{-1}[[p]] \lor m1 \in cdeliver[[p]])$</td>
<td></td>
</tr>
<tr>
<td>/<em>I-13</em>/ $(m1 \rightarrow m2) \in corder \land (p \rightarrow m2) \in cdeliver$</td>
<td>$Broadcast$, $Deliver$</td>
</tr>
<tr>
<td>$\Rightarrow (p \rightarrow m1) \in cdeliver$</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 5. Causal Order Broadcast(Level-1) : Invariants-II

In the last step, to verify that abstract causal order is non-symmetric and non-reflexive, we add invariant $I-10$ and $I-11$ to our model. The B tool generates the proof obligations due to addition of these invariants associated with the events $Broadcast$ and $Deliver$. These proof obligations are discharged interactively without having to add new invariants. By discharging all proof obligations we ensure that this model preserves the causal precedence relationship on the messages and an abstract causal order is transitive, non-symmetric and non-reflexive.

D. Introducing vector clocks : Level-2

In this refinement, we outline how an abstract causal order can be refined by a system of vector clocks. The goals of this refinement are given below.

1. To replace the abstract global variable $corder$ with vector clock rules.
2. To refine the $Broadcast$ event to generate a vector timestamp for messages which realizes the global causal order.
3. To refine the $Deliver$ event to include a mechanism by which an early reception of a message violating the global causal order may be detected at the recipient process.

In a system of vector clocks [22], [23], [27], every process maintains a vector of size $N$ to represent the logical time at that process where $N$ is the total number of processes in the system. Let each process $P_i$ maintains a vector clock $VT_{P_i}$, the clock $VT_{P_i}$ can be defined as a function that assign an event in the process $P_i$ to a vector called vector timestamp. The $VT_{P_i}(i)$ represents a local logical time at $P_i$ while $VT_{P_i}(j)$ represents the process $P_i$’s latest knowledge of the time at process $P_j$. Precisely $VT_{P_i}(j)$ $(i \neq j)$
represents the local time at process $P_j$ when the most recent message was sent from $P_j$ to $P_i$ directly or indirectly. In our model, we used Birman, Schiper and Stephenson’s protocol [8] to update the vector clock of a process broadcasting or delivering a message and to timestamp a message. Important steps of the protocol used in this refinement are outlined below.

I While sending a message $M$ from process $P_i$ to $P_j$, sender process $P_i$ updates its own time ($i^{th}$ entry of vector) by updating $VT_{P_i}(i) := VT_{P_i}(i) + 1$. The message time stamp $VT_M$ of message $M$ is generated as $VT_M(k) := VT_{P_i}(k)$, $\forall k \in (1..N)$, where $N$ is the number of processes in system. Since a process $P_i$ increments its own local time $VT_{P_i}(i)$ only at the time of sending a message, $VT_{P_i}(i)$ indicates number of messages broadcast by process $P_i$.

II The recipient process $P_j$ delays the delivery of message $M$ from $P_i$ until the following conditions are satisfied.

i \quad VT_{P_j}(i) = VT_M(i) - 1

ii \quad VT_{P_j}(k) \geq VT_M(k), \forall k \in (1..N) \land (k \neq i).

The first condition ensures that process $P_j$ has received all but one message sent by process $P_i$. The second condition ensures that process $P_j$ has received all messages received by sender $P_i$ before sending the message $M$.

III The recipient process $P_j$ updates its vector clock $VT_{P_j}$ at the message deliver event of message $M$ as $VT_{P_j}(k) := \text{Max} (VT_{P_j}(k), VT_M(k))$. Therefore, in the vector clock of process $P_j$, $VT_{P_j}(i)$ indicates the number of messages delivered to process $P_j$ sent by process $P_i$.

This refinement (Level-2) consists of four state variables $\text{sender}$, $\text{cdeliver}$, $VT_P$ and $VT_M$. The new state variables $VT_P$ and $VT_M$ respectively represents vector time of a process and the vector time stamp of a message. These variables are typed and initialized as follows.

$$VTP \in \text{PROCESS} \rightarrow (\text{PROCESS} \rightarrow \text{NATURAL})$$

$$VTM \in \text{MESSAGE} \rightarrow (\text{PROCESS} \rightarrow \text{NATURAL})$$

$$VTP := \text{PROCESS} \times \{\text{PROCESS} \times \{0\}\}$$

$$VTM := \text{MESSAGE} \times \{\text{PROCESS} \times \{0\}\}$$

As shown at (7), the variables $VTP$ and $VTM$ are initialized by assigning an array of vector initialized with zero to each process and messages.

The refined specifications of $\text{Broadcast}$ and $\text{Deliver}$ event are given in Fig. 6. A brief description of refinement steps is given in following steps. As shown in the $\text{Broadcast}$ specifications, operations
**Broadcast** \((pp \in \text{PROCESS}, mm \in \text{MESSAGE})\) 

\[
\begin{align*}
\text{WHEN} \quad & nmm \notin \text{dom}(\text{sender}) \\
\text{THEN} \quad & \text{LET} \quad nVTP \quad \text{BE} \quad nVTP = VTP(pp) \land (pp \rightarrow VTP(pp) + 1) \\
& VTM(mm) := nVTP \parallel VTP(pp) := nVTP \quad \parallel \\
& \text{sender} := \text{sender} \cup \{mm \rightarrow pp\} \parallel \\
& \text{cdeliver} := \text{cdeliver} \cup \{pp \rightarrow mm\} \\
\end{align*}
\]

**Deliver** \((pp \in \text{PROCESS}, mm \in \text{MESSAGE})\) 

\[
\begin{align*}
\text{WHEN} \quad & mm \in \text{dom}(\text{sender}) \land \\
& (pp \not\rightarrow mm) \in \text{cdeliver} \land \\
& \forall p. (p \in \text{PROCESS} \land p \not= \text{sender(mm)} \Rightarrow VTP(pp)(p) \geq VTM(mm)(p)) \land \\
& VTP(pp)(\text{sender(mm)}) = VTM(mm)(\text{sender(mm)}) - 1 \\
\text{THEN} \quad & \text{cdeliver} := \text{cdeliver} \cup \{pp \rightarrow mm\} \parallel \\
& VTP(pp) := VTP(pp) \land \\
& (\{q \mid q \in \text{PROCESS} \land VTP(pp)(q) < VTM(mm)(q)\} \land VTM(mm)) \\
\end{align*}
\]

Fig. 6. Refinement with Vector Clocks: Level-2

Involving abstract variable \(corder\) is replaced by the vector rules. It can be noticed that at the time of broadcasting a message \(mm\), process \(pp\) increments its own clock value \(VTP(pp)(pp)\) by one. \(VTP(pp)(pp)\) represents the number of messages sent by process \(pp\). The modified vector timestamp of process is assigned to message \(mm\) giving the vector timestamp of message \(mm\).

As shown in the event **Deliver**, the messages are delivered at a process only if it has delivered all but one message from the sender of that message. The vector time of recipient processes and messages are also compared to ensure that all messages delivered by the sender of the message before sending it, are also delivered at the recipient process. These conditions are included as a guard in the **Deliver** operation. It can be noticed that the guard involving the variable \(corder\) in the abstract model is replaced by the guards involving comparison of the vector timestamps of messages and processes in the refinement.\(^5\), \(^6\)

\(E. \text{Gluing invariants relating abstract causal order and vector rules}\)

The replacement of the operations and the guards involving variable \(corder\) in the abstract model with operations and guards involving vector clock rules in the refinement generate proof obligations. These proof obligations can be discharged interactively using the B tool after three rounds of invariant

\(^5(f \Leftarrow g)\) represents function \(f\) overridden by \(g\).

\(^6(s \Leftarrow f)\) represents function \(f\) is domain restricted by \(s\).

September 18, 2008 DRAFT
strengthening similar to Section III-C. A full set of gluing invariants involving the abstract causal order and the vector clock rules are given in Fig. 7. A brief description of these properties are given below.

- **I-14**: If the vector time of process $P$ is equal or more than the vector time stamp of any sent message $M$ then $P$ must have delivered message $M$.

- **I-15**: For any two messages $m1$ and $m2$ where $m1$ causally precedes $m2$, the vector time stamp of $m1$ is always less than vector time stamp of $m2$.

- **I-16**: Since $VTP(p)(p)$ represents the total number of messages sent by process $p$ and $VTM(m)(p)$ represent number of message delivered to sender of $m$ from process $p$ before sending $m$, the number of messages sent by process $p$ will be greater than or equal to the number of messages delivered to $sender(m)$ from $p$.

- **I-17**: For any two separate processes $p1$ and $p2$, knowledge of $p2$ at $p1$ can not be greater than the knowledge at $p2$ itself.

<table>
<thead>
<tr>
<th>Invariants</th>
<th>Required By</th>
</tr>
</thead>
<tbody>
<tr>
<td>/<em>I-14</em>/ $m \in dom(sender) \land VTP(p1)(p2) \geq VTM(m)(p2)$ $\Rightarrow (p1 \rightarrow m) \in cdeliver$</td>
<td>Broadcast, Deliver</td>
</tr>
<tr>
<td>/<em>I-15</em>/ $(m1 \rightarrow m2) \in corder$ $\Rightarrow VTM(m1)(p) \leq VTM(m2)(p)$</td>
<td>Broadcast, Deliver</td>
</tr>
<tr>
<td>/<em>I-16</em>/ $m \in dom(sender)$ $\Rightarrow VTM(m)(p) \leq VTP(p)(p)$</td>
<td>Broadcast, Deliver</td>
</tr>
<tr>
<td>/<em>I-17</em>/ $p1 \neq p2$ $\Rightarrow VTP(p1)(p2) \leq VTP(p2)(p2)$</td>
<td>Broadcast</td>
</tr>
</tbody>
</table>

Fig. 7. Causal Order Broadcast(Level-2) : Invariants-III

**F. Further refinement of Deliver event : Level-3**

Further refinement of the Deliver event is outlined here stating that instead of updating the whole vector of the recipient process as outlined in the original protocol [8], updating only one value in the vector clock of recipient process corresponding to the sender process is sufficient to realize causally ordered delivery of the messages. In the second refinement (Fig. 6) the vector clock of the recipient process $pp$
is updated as:

\[
VTP(pp) := VTP(pp) \iff \{(q \mid q \in \text{PROCESS} \land
VTP(pp)(q) < VTM(mm)(q) \} < VTM(mm)
\] (8)

The action of event Deliver given at (8) involving update of vector clocks is guarded by following predicates.

\[
\forall p \cdot (p \in \text{PROCESS} \land p \neq \text{sender}(mm) \\
\Rightarrow VTP(pp)(p) \geq VTM(mm)(p))
\] (9)

As shown in the operation at (8), it can be noticed that vector clock of recipient process \( pp \) is updated by the values wherever the values in the message vector are greater, while the guard of the event at (9) indicate that except the sender of message, all values of the message vector must be smaller than recipient process vector. This eventually results in updating only one value of the vector of recipient process which corresponds to the sender of message. Therefore, in the third refinement we replace the operation given at (8) by the simplified operation given at (10). This state that only one value in the vector clock of the recipient process \( pp \) corresponding to the sender process of message is updated.

\[
VTP(pp)(\text{sender}(mm)) := VTM(mm)(\text{sender}(mm))
\] (10)

In this refinement step (Level-3), we observe that proof obligations are generated due to the replacement of the operations of the event Deliver. These proof obligations are automatically discharged by the B prover. The overall proof statistics for this devolvement is outlined in the Table I. It may be noticed that at Level-1 all proof obligations generated due to the refinement checking are discharged automatically by the B tool. The proof obligations that required interaction were generated due to addition of primary invariants outlined in Section III-C.1.

IV. TOTAL CAUSAL ORDER BROADCAST

In this section, we extend the system of a causal order broadcast to a system of total causal order broadcast\(^7\) such that the delivery of the messages also satisfies a total order on the messages in addition to a causal order.

\(^7\)A reliable broadcast that satisfies both causal and total order is also known as causal atomic broadcast.
TABLE I

PROOF STATISTICS - CAUSAL ORDER BROADCAST

<table>
<thead>
<tr>
<th>Machine</th>
<th>Total POs</th>
<th>Completely Automatic</th>
<th>Required Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract Model (L0)</td>
<td>14</td>
<td>14</td>
<td>00</td>
</tr>
<tr>
<td>Refinement-1 (L1)</td>
<td>43</td>
<td>21</td>
<td>22</td>
</tr>
<tr>
<td>Refinement-2 (L2)</td>
<td>47</td>
<td>28</td>
<td>19</td>
</tr>
<tr>
<td>Refinement-3 (L3)</td>
<td>04</td>
<td>04</td>
<td>00</td>
</tr>
<tr>
<td>Overall</td>
<td>108</td>
<td>67</td>
<td>41</td>
</tr>
</tbody>
</table>

A. Mechanism for building a total causal order

Our model is based on Broadcast Broadcast variant [2] of a fixed sequencer algorithm. In this variant, a specific process assumes the role of a sequencer and becomes responsible for building a total order. The protocol consists of first broadcasting a computation message(m) to all destinations including the sequencer, followed by another broadcast of its sequence number through a control message by the sequencer. A process is said to codeliver a message when it is delivered following a causal order. Similarly, a process is said to todeliver a message when it is delivered following a total order. All destination processes todeliver computation messages according to their sequence numbers assigned by the sequencer. Our objective is to verify that abstract causal order is preserved when the messages are todelivered to a process.

The mechanism for implementing a total causal order is outlined through an example in the Fig. 8. In our model, we assume that each process may broadcast a computation message without any restriction and it will eventually be delivered to all processes in the system inclusive of the sender. While broadcasting...
a computation message, an abstract causal order is constructed by the sender. A computation message is first codelivered to all processes inclusive of sequencer. Upon codelivery of a computation message, the sequencer assigns a sequence number to computation message in the order they were codelivered to the sequencer and broadcast its sequencer number through a control message. Upon receiving a control message, a process todelivers a computation message according to their sequence number.

If the broadcast of any two computation messages are not related by a causal precedence relation (parallel messages), causal order broadcast is free to codeliver them in any order at the sequencer. However, the sequencer will assign them the separate sequence numbers guaranteeing that they are delivered to all processes in a total order. Therefore, parallel messages are also delivered to a process in a total order in our model of total causal order broadcast.

1) Outline of the refinement steps: In an incremental development of total causal order broadcast, we begin with an abstract model of a total causal order broadcast. This model is successively refined to a model with vector clocks. A brief outline of each refinement level is outlined below.

L0 This level consists of an abstract model of total causal order broadcast. We verify that this model preserves both causal order and total order properties on the message delivery.

L1 In this refinement, we introduce the notion of computation and control messages. We also outline how the control messages are related to the computation message.

L2 In this refinement, we introduce the notion of vector clocks and sequence numbers. The abstract total order and causal order are replaced by sequence number and vector clocks respectively.

L3 In this refinement, we present a simplification of vector rules for updating the vector clock.

B. Abstract model of total causal order broadcast : Level-0

In order to construct the abstract model of total order broadcast we extend first refinement (Level-1) of causal order broadcast given in Fig. 3. The initial part of the abstract model of total causal order broadcast is given in Fig. 9 as a B machine. Variable sender is used to represent a messages broadcast by a process. The variable cdeliver represents the messages codelivered to the processes following a causal order. Similarly, variable tdeliver represent the messages todelivered to the processes following a total order. The machine also consists of an abstract set variable tordered. This set contains those messages for which sequencer has constructed an abstract total order. Variables causalorder and totalorder are used to represent respectively the abstract causal order and a total order on the messages. To represent the causally ordered delivery of the messages at a process, variable cdeloder is used. A mapping of the form \( (m1 \rightarrow m2) \in cdeloder(p) \) indicate that the process p has codelivered m1 before m2. Similarly, a
Fig. 9. Total Causal Order Broadcast (Level-0) : Initial Part

mapping \((m1 \mapsto m2) \in tdelorder(p)\) indicate that the process \(p\) has \(todelivered\) \(m1\) before \(m2\). It may be noted that a message may have been \(codelivered\) at a process but still waiting for it to be \(todelivered\).

1) Events in the abstract model: The specifications of the events of the abstract model of total causal order broadcast are given in Fig. 10 and Fig. 11. The \textit{Broadcast} event given in the Fig. 10 models the broadcast of a message. It can be noticed that a \textit{causal order} is built by the sender process while broadcasting a message. The event \textit{CausalDeliver} models the event of causally ordered delivery of a message to a process. The guards of the \textit{CausalDeliver} also ensures that a message is \(codelivered\) only once. The following guards of the \textit{CausalDeliver} event ensure that a process \(pp\) causally \textit{codelivers} a message \(mm\) only if it has \(codelivered\) all messages which causally precedes \(mm\).

\[
\forall m.((m \mapsto mm) \in causalorder \Rightarrow (pp \mapsto m) \in cdeliver)
\]

Upon delivery of a message \(mm\) in causal order the variable \(cdelorder\) is also updated so that all messages \(codelivered\) to process \(pp\) are ordered before \(mm\).

The specifications of the events \textit{TOorder} and \textit{TODeliver} are given in Fig. 11. The \textit{TOorder} is an event of constructing an abstract total order on a message. The following guard of this event ensure that a total order for a message \(mm\) is constructed only when it has already constructed total order on all messages that \textit{causally precedes} \(mm\).

\[
\forall m.((m \mapsto mm) \in causalorder \Rightarrow m \in tordered)
\]
**Broadcast** \((pp \in \text{PROCESS}, \ mm \in \text{MESSAGE})\) \(\equiv\)

\[
\text{WHEN } \ mm \in \text{dom}(\text{sender}) \\
\text{THEN} \quad \text{causalorder} := \text{causalorder} \cup (\text{sender}^{-1}\{\{pp\}\} \times \{mm\}) \\
\phantom{\text{causalorder} := \text{causalorder}} \cup (\text{cdeliver}(\{pp\}) \times \{mm\}) \\
\phantom{\text{causalorder} := \text{causalorder} \cup} \text{sender} := \text{sender} \cup \{\text{mm} \rightarrow pp\} \\
\phantom{\text{causalorder} := \text{causalorder} \cup} \text{cdeliver} := \text{cdeliver} \cup \{pp \rightarrow mm\} \\
\phantom{\text{causalorder} := \text{causalorder} \cup} \text{cdelorder}(pp) := \text{cdelorder}(pp) \cup (\text{cdeliver}(\{pp\}) \times \{mm\}) \\
\text{END;}
\]

**CausalDeliver** \((pp \in \text{PROCESS}, \ mm \in \text{MESSAGE})\) \(\equiv\)

\[
\text{WHEN } \ mm \in \text{dom}(\text{sender}) & \\
\phantom{\text{WHEN}} \ (pp \rightarrow mm) \in \text{cdeliver} \wedge \\
\phantom{\text{WHEN}} \ \forall m. (m \in \text{MESSAGE}, (m \rightarrow mm) \in \text{causalorder} \Rightarrow (pp \rightarrow m) \in \text{cdeliver}) \\
\text{THEN} \quad \text{cdeliver} := \text{cdeliver} \cup \{pp \rightarrow mm\} \\
\phantom{\text{WHEN}} \text{cdelorder}(pp) := \text{cdelorder}(pp) \cup (\text{cdeliver}(\{pp\}) \times \{mm\}) \\
\text{END;}
\]

Fig. 10. Total Causal Order Broadcast (Level-0) : Events-I

**TOorder** \((mm \in \text{MESSAGE})\) \(\equiv\)

\[
\text{WHEN } \ mm \in \text{tordered} \wedge \\
\phantom{\text{WHEN}} \ mm \in \text{ran}(\text{cdeliver}) \wedge \\
\phantom{\text{WHEN}} \ \forall m. (m \in \text{MESSAGE}, (m \rightarrow mm) \in \text{causalorder} \Rightarrow m \in \text{tordered}) \\
\text{THEN} \quad \text{tordered} := \text{tordered} \cup (\text{tordered} \times \{mm\}) \\
\phantom{\text{WHEN}} \text{tordered} := \text{tordered} \cup \{mm\} \\
\text{END;}
\]

**TODeliver** \((pp \in \text{PROCESS}, \ mm \in \text{MESSAGE})\) \(\equiv\)

\[
\text{WHEN } \ mm \in \text{tordered} \wedge \\
\phantom{\text{WHEN}} \ (pp \rightarrow mm) \in \text{tdeliver} \wedge \\
\phantom{\text{WHEN}} \ (pp \rightarrow mm) \in \text{cdeliver} \wedge \\
\phantom{\text{WHEN}} \ \forall m. (m \in \text{MESSAGE}, (m \rightarrow mm) \in \text{ttotalorder} \Rightarrow (pp \rightarrow m) \in \text{tdeliver}) \\
\text{THEN} \quad \text{tdeliver} := \text{tdeliver} \cup \{pp \rightarrow mm\} \\
\phantom{\text{WHEN}} \text{tdelorder}(pp) := \text{tdelorder}(pp) \cup (\text{tdeliver}(\{pp\}) \times \{mm\}) \\
\text{END}
\]

Fig. 11. Total Causal Order Broadcast (Level-0): Events-II

A total order for a message \(mm\) is constructed by updating abstract variable \(\text{totaloder}\) as :

\[
\text{totalorder} := \text{totalorder} \cup (\text{tordered} \times \{mm\})
\]

The set \(\text{tordered}\) contains the messages for which a total order has been constructed. The above operation also implies that all messages, for which an abstract total order has been constructed, are now totally
ordered before \( mm \). Later in the refinement (Level-1) we outline that an abstract total order in constructed by the sequencer.

The event \( TODeliver \) models a totally ordered delivery of a message to a process. The guard of this event indicates that a message for which a total order has been constructed, is \( todelivered \) to process only once and if it has also been \( codelivered \) to this process. Upon \( todelivery \) of a message \( mm \), the variable \( tdelorder \) is also updated such that all messages \( todelivered \) to the process \( pp \) are ordered before \( mm \). In the next section, we verify that when the messages are \( todelivered \) to the processes, the causal precedence relationship among them is also preserved.

2) Verification of ordering properties: After developing the abstract model of total causal order, our goal was to verify that this model also preserves the following critical properties. (1) Abstract causal order is preserved when the messages are \( todelivered \) to a process. (2) Both causal and total order properties informally defined in the Section II-A are also preserved. (3) Both abstract causal order and total order are transitive, non-symmetric and non-reflexive. To verify that our model of total causal order broadcast preserves the abstract causal order when the messages are \( todelivered \) to the processes, we need to prove that if the broadcast of any two messages are related by a causal precedence relationship then the same relationship is preserved in the abstract total order. Therefore, we add following to the list of invariant as a primary invariant.

\[
\begin{align*}
m_1 & \in tordered \\
m_2 & \in tordered \\
m_1 \mapsto m_2 & \in causalorder \\
\Rightarrow m_1 \mapsto m_2 & \in totalorder
\end{align*}
\]

This invariant state that if \( m_1 \) precedes \( m_2 \) in abstract causal order and an abstract total order has been constructed on both \( m_1 \) and \( m_2 \) then \( m_1 \) also precedes \( m_2 \) in abstract total order.

| TABLE II |
| Events Code |
| BC | Broadcast | CD | CausalDeliver |
| TO | TOorder | TOD | TODeliver |

The addition of this invariant given at I-9 in Fig. 12 as a primary invariant generates several other proof obligations. In order to discharge these proof obligations we need to add new invariants given as \( Inv-10,11,12 \) to the model. After three rounds of invariant strengthening, we arrive at a set of invariants

September 18, 2008 DRAFT
that is sufficient to discharge all proof obligations. These invariants are outlined in the Fig. 12. The codes for the events are given in the Table II and a brief description of these invariants are given below.

- **I-10**: This invariant states that a message delivered to a process in a total order has also been delivered to that process in a causal order.
- **I-11**: For any two messages \( m_1 \) and \( m_2 \) where \( m_1 \) causally precedes \( m_2 \) and the abstract total order on \( m_2 \) have been constructed, the the abstract total order on \( m_1 \) has also been constructed.
- **I-12**: Each message whose abstract total order has been constructed should have also been code-delivered.

In the next step, to verify that the TotalCausalOrder model also preserves both total order and causal ordering properties, we add a set of invariants given as Invariant-II in Fig. 13 as primary invariants. The **I-13** states the required causal ordering property defined at (2) in Section III-C.1. Addition of this invariants to the model generates proof obligations. Following a similar approach given in the section III we discharge the proof obligations associated with this invariant. To verify that our model also preserves the total ordering properties defined in the section II-A, we add invariant **I-14** as a primary invariant to our model. This invariant state that if a process todelivers any two messages then their delivery order at that process corresponds to their abstract total order. Similarly, in order to prove that both abstract causal order and total order are non-symmetric and non-reflexive, we add invariant **I-15,** **I-16,** **I-17** and **I-18** to our model as primary invariants.

Due to addition of these primary invariants to our model, several proof obligations are generated by the B tool. Following a similar approach of invariant strengthening as outlined in Section III-C.2, after four...
Primary Invariants

/*I-13*/ (m1 ↦ m2) ∈ causalorder ∧ (p ↦ m2) ∈ cdeliver Causal Order
⇒ (m1 ↦ m2) ∈ cdeloder(p)

/*I-14*/ (m1 ↦ m2) ∈ tdelorder(p) Total Order
⇒ (m1 ↦ m2) ∈ totalorder

/*I-15*/ (m1 ↦ m2) ∈ causalorder Causal Order is
⇒ (m2 ↦ m1) ∉ causalorder Non-Symmetric

/*I-16*/ (m1 ↦ m2) ∈ totalorder Total Order is
⇒ (m2 ↦ m1) ∉ totalorder Non-Symmetric

/*I-17*/ m ∈ MESSAGE ⇒ (m ↦ m) ∉ causalorder Causal Order is
Non-reflexive

/*I-18*/ m ∈ MESSAGE ⇒ (m ↦ m) ∉ totalorder Total Order is
Non-reflexive

Fig. 13. Total Causal Order Broadcast (Level-0) : Invariants-II

<table>
<thead>
<tr>
<th>Invariants</th>
<th>Required By</th>
</tr>
</thead>
<tbody>
<tr>
<td>/<em>I-19</em>/ (p ↦ m1) ∈ tdeliver ∧ (p ↦ m2) ∈ tdeliver TOD</td>
<td></td>
</tr>
<tr>
<td>∧ m2 ∈ ran(tdeliver)⇒ (m1 ↦ m2) ∈ totalorder</td>
<td></td>
</tr>
<tr>
<td>/<em>I-20</em>/ m1 ∈ ran(tdeliver) ∧ m2 ∈ ran(tdeliver) TO,CD,TOD</td>
<td></td>
</tr>
<tr>
<td>∧ (m2 ↦ m1) ∈ totalorder⇒ (m1 ↦ m2) ∈ totalorder</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 14. Total Causal Order Broadcast (Level-0) : Invariants-III

rounds of invariant strengthening we arrive at a set of invariant given in Fig. 14 which were sufficient
to discharge all proof obligations. A brief description of the properties is given below.

- I-19 : If a process p has delivered m1 and but not m2, and if m2 was delivered to at least one process
  anywhere in the system then m1 precedes m2 in total order.

- I-20 : If two messages m1 and m2 has been delivered anywhere in the system then a total order
  exist among them, such that, either m1 precedes m2 or m2 precedes m1 in total order.

- I-21 : If a process p has delivered two message m1 and m2 then either m1 precedes m2 or m2
precedes \emph{m1} in total order.

- \textbf{I-22} : Given two processes \emph{p1} and \emph{p2}, then for any two messages \emph{m1} and \emph{m2} if the process \emph{p2} has delivered both messages and \emph{p1} has delivered \emph{m1} but not \emph{m2} then \emph{m1} precedes \emph{m2} in total order.

<table>
<thead>
<tr>
<th>Invariants</th>
<th>Required By</th>
</tr>
</thead>
<tbody>
<tr>
<td>/<em>I-23</em>/ ((m1 \rightarrow m2) \in causalorder \land (m2 \rightarrow m3) \in causalorder \Rightarrow (m1 \rightarrow m3) \in causalorder)</td>
<td>Causal Order is Transitive</td>
</tr>
<tr>
<td>/<em>I-24</em>/ ((m1 \rightarrow m2) \in totalorder \land (m2 \rightarrow m3) \in totalorder \Rightarrow (m1 \rightarrow m3) \in totalorder)</td>
<td>Total Order is Transitive</td>
</tr>
<tr>
<td>/<em>I-25</em>/ ((m1 \rightarrow m2) \in causalorder \land (p \rightarrow m2) \in cdeliver \Rightarrow (p \rightarrow m1) \in cdeliver)</td>
<td>BC, CD, TO, TOD</td>
</tr>
<tr>
<td>/<em>I-26</em>/ ((m1 \rightarrow m2) \in totalorder \land (p \rightarrow m2) \in tdeliver \Rightarrow (p \rightarrow m1) \in tdeliver)</td>
<td>TO, TOD</td>
</tr>
</tbody>
</table>

Fig. 15. Total Causal Order Broadcast (Level-0) : Invariants-IV

In the next step we verify that abstract model of total causal order also satisfies the transitivity properties on both abstract causal order and total order. Therefore, we add \textbf{I-23} and \textbf{I-24} given in Fig. 15 to our model as primary invariants. The addition of these invariants generate new proof obligations which are discharged by adding \textbf{I-25} and \textbf{I-26}. A brief description of these properties are given below.

- \textbf{I-23} : An abstract causal order is transitive.
- \textbf{I-24} : An abstract total order is transitive.
- \textbf{I-25} : For two messages \emph{m1} and \emph{m2}, if \emph{m1} causally precedes \emph{m2} and process \emph{p} has \emph{codelivered} the message \emph{m2} then \emph{p} has also \emph{codelivered} the message \emph{m1}.
- \textbf{I-26} : For two messages \emph{m1} and \emph{m2}, if \emph{m1} precedes \emph{m2} in \emph{total order} and process \emph{p} has \emph{todelivered} the message \emph{m2} then \emph{p} has also \emph{todelivered} \emph{m1}.

\textbf{C. Introducing control messages : Level-1}

In this section, we outline the first refinement of the abstract model of total causal order broadcast. A notion of \textit{sequencer} is introduced in this refinement. The sequencer is defined as a constant where a process is assigned as sequencer non-deterministically. The notion of \textit{computation} and \textit{control} messages is also introduced in this refinement. This refinement consists of following state variables typed as follows,
computation ⊆ MESSAGE
control ⊆ MESSAGE
messcontrol ∈ control ↔ computation
receive ∈ PROCESS ↔ control

The set variables control and computation are used to represent computation and control messages. The set control also represent the control messages sent by the sequencer. The variable messcontrol is a partial injective function which defines relationship between a computation message and its control message. A mapping \((m_1 \mapsto m_2) \in \text{messcontrol}\) indicate that message \(m_1\) is the control message related to the computation message \(m_2\). Since messcontrol is defined as a partial injective function, it also implies that there can be only one control message for each computation message and vice-versa. The set \(\text{ran(messcontrol)}\) contains the computation messages for which control messages has been sent by the sequencer. The variable receive is used to represent the control messages received by a process. A mapping \(p \mapsto m \in \text{receive}\) indicate that process \(p\) has received a control message \(m\). Specifications of the events in this refinement are given in Fig. 16 and 17.

### Broadcast

\[
\text{Broadcast}(pp \in \text{PROCESS}, mm \in \text{MESSAGE}) \equiv \\
\text{WHEN } mm \notin \text{dom(sender)} \\
\text{THEN } \begin{align*}
\text{causalorder} &\leftarrow \text{causalorder} \cup (\text{sender}^{-1}([pp] \times [mm]) \\
&\cup (\text{cdeliver}([pp] \times [mm]))) \\
\text{sender} &\leftarrow \text{sender} \cup \{mm \mapsto pp\} \\
\text{cdeliver} &\leftarrow \text{cdeliver} \cup \{pp \mapsto mm\} \\
\text{computation} &\leftarrow \text{computation} \cup \{mm\}
\end{align*} \\
\text{END;}
\]

### CausalDeliver

\[
\text{CausalDeliver}(pp \in \text{PROCESS}, mm \in \text{MESSAGE}) \equiv \\
\text{WHEN } mm \in \text{dom(sender)} \land \\
mm \in \text{computation} \land \\
pp \mapsto mm \in \text{cdeliver} \land \\
\forall m.(m \in \text{computation} \land (m \mapsto mm) \in \text{causalorder}) \\
\Rightarrow (pp \mapsto m) \in \text{cdeliver}
\text{THEN } \text{cdeliver} \leftarrow \text{cdeliver} \cup \{pp \mapsto mm\} \\
\text{END;}
\]

Fig. 16. Total Causal Order Broadcast (Level-1) : Events-I

As shown in the Fig. 16, while broadcasting a message \(mm\), it is also casted as computation message. A new event ReceiveControl is introduced in this refinement. This event models receiving a control message by a process. It may be noted that a process must receive a control message before it todelivers...
TOorder \((pp \in \text{PROCESS}, mm \in \text{MESSAGE}, mc \in \text{MESSAGE})\) \equiv \\
\text{WHEN} \quad pp = \text{sequencer} \land \\
\hspace{1cm} mm \in \text{computation} \land \\
\hspace{1cm} mc \notin \text{control} \land \\
\hspace{1cm} mm \notin \text{ran(messcontrol)} \land \\
\hspace{1cm} (pp \rightarrow mm) \in \text{cdeliver} \land \\
\hspace{1cm} \forall m. (m \in \text{computation} \land (m \rightarrow mm) \in \text{causalorder}) \implies m \in \text{ran(messcontrol)} \\
\text{THEN} \quad \text{control} := \text{control} \cup \{mc\} \parallel \\
\hspace{1cm} \text{messcontrol} := \text{messcontrol} \cup \{mc \rightarrow mm\} \parallel \\
\hspace{1cm} \text{totalorder} := \text{totalorder} \cup (\text{ran(messcontrol)} \times \{mm\}) \\
\text{END:} \\
\text{ReceiveControl}(pp \in \text{PROCESS}, mc \in \text{MESSAGE}) \equiv \\
\text{WHEN} \quad mm \in \text{control} \land \\
\hspace{1cm} (pp \rightarrow mc) \notin \text{receive} \\
\text{THEN} \quad \text{receive} := \text{receive} \cup \{pp \rightarrow mc\} \\
\text{END} \\
\text{TODeliver} \((pp \in \text{PROCESS}, mm \in \text{MESSAGE})\) \equiv \\
\text{WHEN} \quad mm \in \text{computation} \land \\
\hspace{1cm} (pp \rightarrow \text{messcontrol}^t(mm)) \notin \text{receive} \land \\
\hspace{1cm} (pp \rightarrow mm) \in \text{cdeliver} \land \\
\hspace{1cm} (pp \rightarrow mm) \notin \text{tdeliver} \land \\
\hspace{1cm} \forall m. (m \in \text{computation} \land (m \rightarrow mm) \in \text{totalorder}) \implies (pp \rightarrow m) \in \text{tdeliver} \\
\text{THEN} \quad \text{tdeliver} := \text{tdeliver} \cup \{pp \rightarrow mm\} \\
\text{END} \\

Fig. 17. Total Causal Order Broadcast (Level-1): Events-II

corresponding computation message. This requirement is modelled in the specification of the TOdeliver event in the Fig. 17.

As shown in the specifications of the event TOorder, a total order for a message \(mm\) is constructed by the sequencer. The abstract variable \(tordered\) is also replaced by the set \(\text{ran(messcontrol)}\) in this refinement. The abstract variable \(tordered\) contains the messages for which total order has been constructed while the set \(\text{ran(messcontrol)}\) is set of computation message for which control message has been sent by the sequencer. The replacement of the variables and introduction of new event generate new proof obligations. These proof obligations are discharged by adding following invariants given in Fig. 18.

D. Introducing vector clocks and the sequence numbers : Level-2

In this refinement we introduce the notion of vector clocks and sequence numbers. The abstract variables causalorder and totalorder in this refinement are replaced with the vector clock rules and sequence
numbers respectively. To implement total ordering we also introduce variables \textit{seqno} and \textit{counter}.

1) Events in the second refinement: The events of the second refinement are given in the Fig. 19 and Fig. 20. It can be noticed that the events involving abstract variable \textit{causalorder} are replaced by the vector rules. Similarly, the events involving abstract variable \textit{totalorder} are replaced by sequence numbers (\textit{seqno}). The the specification of the event \textit{ReceiveControl} are same as at Level-1.

\begin{verbatim}
Broadcast(pp \in PROCESS, mm \in MESSAGE) =
  WHEN mm \notin \text{dom}(sender)  
  THEN LET nVTP BE nVTP = VTP(pp) \leq \{ pp \mapsto VTP(pp)(pp)+1 \}
     IN VTM(mm) := nVTP \quad VTP.pp := nVTP 
    END \quad ||
     sender := sender \cup \{ mm \mapsto pp \} \quad ||
     cdeliver := cdeliver \cup \{ pp \mapsto mm \} \quad ||
     computation := computation \cup \{ mm \} 
    END;

CausalDeliver (pp \in PROCESS, mm \in MESSAGE) =
  WHEN mm \in \text{dom}(sender) \land
     mm \in \text{computation} \land
     (pp \mapsto mm) \in \text{cdeliver} \land
     \forall p (p \in PROCESS \land p \neq \text{sender(mm)} \implies VTP(pp)(p) \geq VTM(mm)(p)) \land
     VTP(pp)(\text{sender(mm)}) = VTM(mm)(\text{sender(mm)})-1
  THEN cdeliver := cdeliver \cup \{ pp \mapsto mm \} \quad ||
     VTP(pp) := VTP(pp) \land
     (\{q | q \in PROCESS \land VTP(pp)(q) < VTM(mm)(q)\} \land VTM(mm)) 
    END;
\end{verbatim}

Fig. 19. Total Causal Order Broadcast (Level-2) : Events-I

The variable \textit{seqno} is used for building a total order on the computation messages. In the refined specification of event \textit{TOorder}, it can be noticed that operation involving the abstract \textit{totalorder} is replaced by an operation containing variable \textit{seqno} and \textit{counter}. The counter is incremented each time a control message is sent and it is assigned to the control messages. The guards of the event \textit{TOdeliver} are strengthened in this refinement. Consider following guard of the event \textit{TOdeliver} involving abstract...
TOorder \( (pp \in \text{PROCESS}, mm \in \text{MESSAGE}, mc \in \text{MESSAGE}) \) \( \equiv \)

\[
\text{WHEN } \begin{align*}
& pp = \text{sequencer} \land \\
& mm \in \text{computation} \land \\
& mc \notin \text{control} \land \\
& mm \notin \text{ran}\text{(messcontrol)} \land \\
& pp \rightarrow mm \in \text{tdeliver} \land \\
& \forall (m,p) \cdot (p \in \text{PROCESS} \land m \in \text{MESSAGE} \land m \in \text{computation} \\
& \land \ VTM(m)(p) \leq VTM(mm)(p) \Rightarrow m \in \text{ran}\text{(messcontrol)} \\
\end{align*}
\]

\[
\text{THEN } control := control \cup \{mc\} \\
\text{messcontrol} := \text{messcontrol} \cup \{mc \rightarrow mm\} \\
\text{LET } ncount \text{ BE } ncount = \text{counter} + 1 \\
\text{IN } \text{counter} := ncount \parallel \text{seqno}(mm) := ncount \text{ END}
\]

END;

TODeliver \( (pp \in \text{PROCESS}, mm \in \text{MESSAGE}) \) \( \equiv \)

\[
\text{WHEN } \begin{align*}
& mm \in \text{computation} \land \\
& (pp \rightarrow \text{messcontrol}^2(mm)) \in \text{receive} \land \\
& (pp \rightarrow mm) \in \text{tdeliver} \land \\
& (pp \rightarrow mm) \notin \text{tdeliver} \land \\
& \forall m. (m \in \text{computation} \land (\text{seqno}(m) < \text{seqno}(mm)) \\
& \Rightarrow (pp \rightarrow m) \in \text{tdeliver})
\end{align*}
\]

\[
\text{THEN } \text{tdeliver} := \text{tdeliver} \cup \{pp \rightarrow mm\}
\]

END

Fig. 20. Total Causal Order Broadcast (Level-2) : Events-II

totalorder.

\[
\forall m \cdot (m \in \text{computation} \land (m \rightarrow mm \in \text{totalorder}) \\
\Rightarrow (pp \rightarrow m) \in \text{tdeliver}) \tag{11}
\]

\[
\forall m \cdot (m \in \text{computation} \land (\text{seqno}(m) < \text{seqno}(mm)) \\
\Rightarrow (pp \rightarrow m) \in \text{tdeliver}) \tag{12}
\]

Note that the guard at (11) at Level-1 is replaced by (12) in this refinement (Level-2). This states that the process has todelivered all computation messages whose sequence number is less than the sequence number of computation message \( mm \).

2) \textit{Gluing invariants}: The replacement of guards involving abstract causal order by vector clocks and abstract total order by sequence number generate new proof obligations. These proof obligations requires us to construct and add new invariants relating abstract causalorder and totalorder with vector clock rules and sequence numbers to our model. After three rounds of invariant strengthening we arrive at a
set of gluing invariants that are sufficient to discharge all proof obligations. These invariants are given in the Fig. 21. A brief description of these properties are given below.

- \textbf{I-29} : If the vector time at a process $P$ is equal or more than vector time stamp of any sent message $M$ then $P$ must have \textit{codelivered} the message $M$.
- \textbf{I-30} : For any two messages $m_1$ and $m_2$ where $m_1$ causally precedes $m_2$, the vector time stamp of $m_1$ is less than vector time stamp of $m_2$.
- \textbf{I-31} : Since $VTP(p)(p)$ represent total number of message sent by a process $p$ and $VTM(m)(p)$ represent number of message received by the sender of $m$ from process $p$ before sending $m$, the number of messages sent by process $p$ will be greater than or equal to the number of messages received by the $sender(m)$ from $p$.
- \textbf{I-32} : If any two computation messages $m_1$ and $m_2$ are in \textit{totalorder} then sequence number of $m_1$ is less than the sequence number of $m_2$.

<table>
<thead>
<tr>
<th>Invariants</th>
<th>Required By</th>
</tr>
</thead>
<tbody>
<tr>
<td>/<em>I-29</em>/ $m \in dom(sender) \times VTP(p1)(p2) \geq VTM(m)(p2)$ $\Rightarrow (p1 \rightarrow m) \in cdeliver)$</td>
<td>BC,CD,TO</td>
</tr>
<tr>
<td>/<em>I-30</em>/ $(m1 \rightarrow m2) \in causalorder$</td>
<td>BC,CD</td>
</tr>
<tr>
<td>$\Rightarrow VTM(m1)(p) \leq VTM(m2)(p))$</td>
<td></td>
</tr>
<tr>
<td>/<em>I-31</em>/ $m \in dom(sender) \Rightarrow VTM(m)(p) \leq VTP(p)(p)$</td>
<td>BC,CD</td>
</tr>
<tr>
<td>/<em>I-32</em>/ $(m1 \rightarrow m2) \in totalorder \Rightarrow seqno(m1) &lt; seqno(m2)$</td>
<td>TO,TOD</td>
</tr>
</tbody>
</table>

Fig. 21. Total Causal Order Broadcast (Level-2) : Invariants-VI

\textbf{E. Further refinement (Level-3) and proof statistics}

In these refinements, similar to the steps outlined in the Section III-F, we simplify the operations of \textit{CausalDeliver} event (Fig. 19) that update vector clock of a recipient process. The replacement of the guards generate proof obligations that are discharged automatically by the B tool. The overall proof statistics for the development of a system of a total causal order is given in Table III.
V. RELATED WORK

Though there exists vast literature on distributed algorithms and protocols covering several aspects of transactions, group communication and replicated databases, only a few papers actually specify the abstract problem that they solve. Applications of proof based formal methods for precise definition of the problems and verification of the correctness of their solutions is still an important problem in distributed systems. In this section, we compare our approach with the other significant work on the application of formal methods.

The input/output (I/O) automaton model [28], [29], developed by Lynch and Tuttle, is a labelled transition system model for components in asynchronous concurrent systems. In [30], I/O automata are used for formal modelling and verification of a sequentially consistent shared object system in a distributed network. In order to keep the replicated data in a consistent state, a combination of total order multicast and point to point communication is used. In [31], I/O automata is used to express lazy database replication. The authors present an algorithm for lazy database replication and prove the correctness properties relating performance and fault tolerance. In [4], [32] the specification for group communication primitives are presented using I/O automata under different conditions such as partitioning among the group and dynamic view oriented group communication. The proof method supported in this method for reasoning about the system involves invariant assertions. An invariant assertion is defined as a property of the state of a system that is true in all execution. A series of invariants relating state variables and reachable states are proved by hand using the method of induction.

In [33], a formal method is proposed to prove the total and causal order of multicast protocols. The formal results are provided in the paper that can be used to prove whether an existing system has the required property or not. Their solutions are based on the assumption that a total order is built using the
service provided by a causal order protocol. The proof of correctness of the results are done by hand. In a similar work in [34], meta properties are used to express total order broadcast algorithm. In [35], $Z$ is used to specify formally a transaction decomposition in a database system and for expressing various models. A notion of semantic histories used to formulate and prove necessary properties, and reason about interleaving with other transactions. The authors have done all analysis by hand. They also highlight the need for powerful tool support to discharge proof obligations.

A refinement based approach to developing distributed systems in B is outlined in [36]. The correspondence between the action based formalism and the abstract B machines is outlined in this work. The action system formalism [37] is a state based approach to distributed computing. An action system models a reactive system with guarded actions on state variables. In [36], the author outlines how the reactive refinement and decomposition of action systems can be applied to abstract machines and how this approach is related to step wise refinement in B. The refinement approach has been applied to the development of a secure communication system [38]. The aim was to carry out a development from initial abstract specifications of security services to a detailed design in the refinement steps. The authors have also demonstrated an effective way to combine B and CSP specifications. In [39] important contributions are made towards development of a refinement rule which allows actions to be introduced in a refinement step and a decomposition rule which allows a system model to be decomposed into parallel subsystems. Use of refinement and decomposition rules in the development of telecommunications systems is outlined in [40]. Other important work carried out using the refinement approach include the Mondex purse system [18] in Event-B, verification of the IEEE 1394 tree protocol distributed algorithm [41], development of a secure communication system [38], development of a train system [42], verification of one copy equivalence criterion in a distributed database system [9]. The case study on Mondex illustrates modelling strategies and the guidelines to achieve a high degree of automatic proofs.

VI. Conclusion

Distributed algorithms can be deceptive. An algorithm that looks simple may have complex execution paths and allow unanticipated behavior. Rigorous reasoning about the algorithms is required to ensure that an algorithm achieves what it is supposed to do [1]. In this paper, we outlined a refinement approach using Event-B for the formal development of the models of distributed systems. We have outlined the abstract specifications of causal order broadcast and verify how an abstract causal order can correctly be implemented by a system of vector clocks. We also found that instead of updating the whole vector of the recipient process as outlined in the original protocol [8], updating only one value in the vector clock
of recipient process corresponding to the sender process is sufficient to realize causally ordered delivery of the messages. In a similar development, we presented an incremental development of a model of total causal order broadcast that satisfies both causal and a total order. In the refinement, we verify that a total order can be built by using the services of vector clocks alone, replacing the need for explicit sequence numbers to be generated by the sequencer.

Instead of model checking, proving theorems by hand or proving correctness of the trace behavior, our approach consists of defining problem in the abstract model and introducing solutions or design details in the refinement steps. Through refinement checking we verify that the models in the refinement are valid refinement of abstract models. There exist several industrial level B tools to aid formal development of distributed systems. We used the Click'n’Prove B tool the proof management. This tool generates the proof obligations due to refinement and consistency checking, factorizes complex proof obligations in to relatively simpler proofs and helps discharge proof obligations by the use of automatic and interactive prover. In our case studies approximately 60% of the proof obligations were discharged using automatic prover. The proof obligations generated by the B tool also help discovering new system invariants. The proofs and the invariants help to precisely understand why a design decision or a solution proposed in the refinement is correct.

REFERENCES


