Application of Event B to Global Causal Ordering for Fault Tolerant Transactions

Divakar Yadav and Michael Butler

Declarative Systems and Software Engineering
School of Electronics and Computer Science
University of Southampton
Southampton SO17 1BJ UK
Event B

- B Method is a proof based formal method developed by Abrial.
- Event B is event driven approach used together with B Method.
- Event B provides complete framework for developing mathematical model of distributed algorithms by
  - Rigorous description of problem.
  - Gradually introducing solution in refinement steps.
  - Verification of correctness of solution by discharging proof obligations.
- Atelier B, Click’n’Prove, B Toolkit provides support for discharge of proof obligation through automatic and interactive prover.
Fault Tolerant Transactions
Some issues on our ongoing work

- Distributed System is a collection of autonomous computers spatially separated.

- Fragmentation and Replication of data is a key issue in distributed database.

- Synchronous replication techniques require that all replica are updated before updating distributed transaction commits.

- Read One Write All (ROWA) based synchronous replication requires transaction to read one copy and write all copies.

- Fault Tolerance may be achieved by either masking failures or by following well defined behaviour suitable for recovery.
Synchronous Replication
Read One Write All (ROWA)

Sites contains the replica of data object.
Synchronous Replication
Read One Write All (ROWA)

- Initial value of data is U.
Synchronous Replication
Read One Write All (ROWA)

Transaction $T_i$ is submitted at site $S_i$. 
Site $S_i$ sends messages to participating sites.
Synchronous Replication
Read One Write All (ROWA)

Sub transactions of $T_i$ starts at participating sites
Distributed Transaction $T_i$ commits only if all Sub transactions commits.
Synchronous Replication
Read One Write All (ROWA)

Distributed Transaction $T_i$ Aborts if *Any* Sub transactions aborts.
If Distributed Transaction $T_i$ Aborts, it aborts at all sites.

$\Rightarrow$ None of replica is updated.
Synchronous Replication
Read One Write All (ROWA)

If Distributed Transaction $T_i$ Commits, it commits at all sites.

$\Rightarrow$ All replicas are updated.
From now onwards….

Application of Event B to

- Broadcast messaging system.
- Buffering of messages.
- Abstract model of causal order.
- Globally ordered delivery of messages.
- Implementation through vector clocks.
Processes communicate by broadcasting of messages.

No loss or duplication of message.

Messages are delivered after arbitrary delays.
Broadcast Messaging

SETS PROCESS; MESSAGE

VARIABLES sender, receive

INITIALISATION

sender := ∅ || receive := ∅

INVARIANT

sender ∈ MESSAGE ℮ PROCESS
receive ∈ PROCESS ℮ MESSAGE
(p ↦ m) ∈ receive ⇒ m ∈ dom(sender)
(p ↦ m) ∈ receive ⇒ p ≠ sender(m)

OPERATIONS

Send(pp, mm) ≡
SELECT mm ∈ dom(sender)
THEN sender := sender ∪ {mm ↦ pp}
END;

Receive (pp, mm) ≡
SELECT mm ∈ dom(sender)
∧ (pp ↦ mm) ∉ receive
∧ pp ≠ sender(mm)
THEN receive := receive ∪ {pp ↦ mm}
END
Happened Before Relation

- The *happened before* relation captures *causal dependency* between various events occurring in a process.

- *Message Send* and *Message Receive* are message events.

- Event A and B are *causally related* if either $A \rightarrow B$ or $B \rightarrow A$.

- Event A and B are *concurrent* ($A \parallel B$) if $A \not\rightarrow B$ and $B \not\rightarrow A$.

- **Transitivity**: $A \rightarrow B \land B \rightarrow C \Rightarrow A \rightarrow C$
Some Observations

- $E_1 \rightarrow E_2$
- $E_2 \rightarrow E_4$
- $E_1 \rightarrow E_2 \land E_2 \rightarrow E_4 \Rightarrow E_1 \rightarrow E_4$
- $E_1 \parallel E_3$
Causal Ordering of Messages

Some Observations

Message $M_1$ (dash-dot arrow) shows violation of global causal ordering.
Message $M_1$ shows violation of global causal ordering.
Causal Ordering of Messages to Broadcast System

Some Observations

\begin{align*}
\text{M1} &\rightarrow \text{M2} \\
\text{M2} &\rightarrow \text{M3} \\
\text{M1} &\rightarrow \text{M2} \land \text{M2} \rightarrow \text{M3} \implies \text{M1} \rightarrow \text{M3}
\end{align*}
Abstract Model of Causal Order
First Refinement

VARIABLES sender, receive, order

INVARIANT

order ∈ MESSAGE ↔ MESSAGE

If M₁ → M₂ and P has received M₂, then P must have received M₁

If M₁ → M₂ and P has sent M₂, then P must have sent or received M₁

order is transitive

INITIALISATION sender := ∅ || receive := ∅ || order := ∅
Abstract Model of Causal Order
First Refinement

VARIABLES sender, receive, order

INVARIANT

order ∈ MESSAGE ↔ MESSAGE

(m1→m2) ∈ order ∧ (p→m2) ∈ receive ∧ p ≠ sender(m1) ⇒ (p→m1) ∈ receive

(m1→m2) ∈ order ∧ (m2→p) ∈ sender ⇒ ((m1→p) ∈ sender ∨ (p→m1) ∈ receive)

(m1 → m2) ∈ order ∧ (m2 → m3) ∈ order ⇒ (m1 → m3) ∈ order

INITIALISATION sender := ∅ || receive := ∅ || order := ∅
Operations

Operations

Operations

Send \((pp,mm)\) \equiv \text{SELECT } mm \in \text{dom}(sender) \text{ THEN order := order } \cup \text{ (sender-\{pp\} } \ast \{mm\}) \cup \text{ (receive[\{pp\} } \ast \{mm\})) || \text{ sender := sender } \cup \{mm \mapsto pp\} \text{ END;}

Receive \((pp,mm)\) \equiv \text{SELECT } mm \in \text{dom}(sender) \text{ THEN } \forall m. (m \in \textit{MESSAGE} \land \forall m. (m \mapsto mm) \in \text{order} \land \forall m. (m \mapsto mm) \in \text{order} ) \text{ THEN receive := receive } \cup \{pp \mapsto mm\} \text{ END}
Buffering of Messages
Second Refinement

- To ensure *globally ordered delivery* of messages at a recipient process, early message need be buffered.

- For any two message M1, M2 where M1 is ordered before M2 (M1 → M2), if M2 *arrives* early at a process then M2 is *buffered* until M1 is received.
Buffering of Messages
Second Refinement

SETS PROCESS ; MESSAGE VARIABLES sender , receive , order , buffer

INITIALISATION sender := Ø || receive := Ø || order := Ø || buffer := Ø

INTRODUCING A NEW EVENT Arrive

INARIANT

buffer ∈ PROCESS ↔ MESSAGE

ran(buffer) ⊆ dom(sender)

ran(receive) ∩ ran(buffer) = Ø

OPERATIONS

Arrive (pp,mm) ≜ SELECT mm ∈ dom(sender)

∧ (pp ↦ mm) ∈ buffer

∧ (pp ↦ mm) ∈ receive

∧ pp ≠ sender(mm)

THEN

buffer := buffer ∪ {pp ↦ mm}

END ;
Logical Clocks : Vector Clock

- Vector Clock uses a vector of Integers of size $N$, where $N$ is number of processes in system.

- Process $P_i$ maintains a vector clock $VT_i$.

- $VT_i[i]$ is process $P_i$’s own logical time.

- $VT_i[j]$ is process $P_i$’s best knowledge of time at process $P_j$.

- Proposed by Fidge and Mattern and based on Lamport’s scalar clocks
Applying Vector Clocks
to Broadcast System
Applying Vector Clocks
to Broadcast System
Applying Vector Clocks

to Broadcast System

P1
0,0,0

1,0,0 → 2,0,0

1,0,0 → 2,0,0

2,0,0 → 2,1,0

P2
0,0,0

1,0,0 → 2,0,0

2,0,0 → 2,1,0

P3
0,0,0

1,0,0 → 2,0,0

2,0,0 → 2,1,0
Applying Vector Clocks to Broadcast System
Some Observations

- \( VT_i[i] \) indicates number of messages sent by process \( Pi \).

- \( VT_j[i] \) indicates number of messages received by process \( Pj \) sent by process \( Pi \).
Applying Vector Clocks to Ensure Globally Ordered Delivery of Messages

- Process $P_i$ broadcasts a message $M$.

- A recipient process $P_j$ delays the delivery of message $M$ until the following conditions are satisfied:
  
  $VT_j[i] = VT_M[i] - 1$
  
  $VT_j[k] \geq VT_M[k], \forall k \in (1..N) \land (k \neq i)$
Applying Vector Clocks to Ensure Globally Ordered Delivery of Messages

Message $M_2$ arrives early at $P_3$. 

- Message $M_1$ arrives at $P_1$.
- Message $M_2$ arrives at $P_3$.

Nodes:
- $P_1$: $(1,0,0)$
- $P_2$: $(1,0,0)$
- $P_3$: $(2,0,0)$
Applying Vector Clocks to Ensure Globally Ordered Delivery of Messages

Message $M_2$ arrives early at $P_3$. 

- Path $P_1$: $0,0,0$ to $1,0,0$ to $1,1,0$
- Path $P_2$: $0,0,0$ to $1,0,0$ to $1,1,0$
- Path $P_3$: $0,0,0$ to $0,0,0$ (Self message)

Diagram shows the vector clocks at each node, with arrows indicating the sequence of messages.
Applying Vector Clocks
Third Refinement

Introducing a new variables VTP and VTM

SETS

PROCESS ; MESSAGE

VARIABLES

sender, receive, order, buffer, VTP, VTM

INVARIANT

\[ VTP \in \text{PROCESS} \rightarrow (\text{PROCESS} \rightarrow \mathbb{N}) \]

\[ \wedge VTM \in \text{MESSAGE} \rightarrow (\text{PROCESS} \rightarrow \mathbb{N}) \]
Applying Vector Clocks
Third Refinement

Refinement of Operation Send

Send (pp,mm) \triangleq
SELECT mm \notin \text{dom}(sender)
THEN
order := order \cup (sender\sim\{pp\} \times \{mm\})
\cup (receive\{pp\} \times \{mm\})
\parallel sender := sender \cup \{mm \mapsto pp\}
END;

Send(pp,mm) \triangleq
SELECT mm \notin \text{dom}(sender)
\land VTP(pp)(pp) \geq 0
\land VTP(pp)(pp) < \text{MAXINT}
THEN
LET nVTP BE
nVTP = VTP(pp) \leftarrow \{ pp \mapsto VTP(pp)(pp)+1\}
IN VTM(mm) := nVTP \parallel VTP(pp) := nVTP
\parallel sender := sender \cup \{mm \mapsto pp\}
END;
Applying Vector Clocks

Third Refinement

Refinement of Operation Receive

Receive (pp,mm) \triangleq 
\begin{align*}
\text{SELECT} & \quad \text{mm} \in \text{dom(sender)} \land (pp \mapsto \text{mm}) \notin \text{receive} \land pp \neq \text{sender(mm)} \\
& \quad \land \forall m. (m \in \text{MESSAGE} \land (m \mapsto \text{mm}) \in \text{order} \land pp \neq \text{sender(m)} \implies (pp \mapsto m) \in \text{receive}) \\
\text{THEN} & \quad \text{receive} := \text{receive} \cup \{pp \mapsto \text{mm}\} \quad \text{||} \quad \text{buffer} := \text{buffer} - \{pp \mapsto \text{mm}\} \\
\text{END}
\end{align*}

Recieve(pp,mm) \triangleq 
\begin{align*}
\text{SELECT} & \quad \text{mm} \in \text{dom(sender)} \land (pp \mapsto \text{mm}) \notin \text{receive} \land pp \neq \text{sender(mm)} \\
& \quad \land (pp \mapsto \text{mm}) \in \text{buffer} \\
& \quad \land \forall p. (p \in \text{PROCESS} \land p \neq \text{sender(mm)} \implies \text{VTP}(pp)(p) > \text{VTM(mm)(p)}) \\
& \quad \land \text{VTP}(pp)(\text{sender(mm)}) = \text{VTM}(\text{mm}(\text{sender(mm)})) - 1 \\
\text{THEN} & \quad \text{receive} := \text{receive} \cup \{pp \mapsto \text{mm}\} \quad \text{||} \quad \text{buffer} := \text{buffer} - \{pp \mapsto \text{mm}\} \\
& \quad \text{||} \quad \text{VTP}(pp) := \text{VTP}(pp) \sqsubseteq (\{q \mid q \in \text{PROCESS} \land \text{VTP}(pp)(q) < \text{VTM(mm)(q)}\} \sqsubseteq \text{VTM(mm)}) \\
\text{END}
\end{align*}
Applying Vector Clocks
Third Refinement

INVARIANT

\[ \forall m_1, m_2, p. (m_1 \in \text{MESSAGE} \land m_2 \in \text{MESSAGE} \land p \in \text{PROCESS} \]
\[ \land (m_1 \rightarrow m_2) \in \text{order} \Rightarrow \text{VTM}\left(m_1(p)\right) \leq \text{VTM}\left(m_2(p)\right) ) \]

\[ \land \forall p_1, m, p. (p_1 \in \text{PROCESS} \land p \in \text{PROCESS} \land m \in \text{MESSAGE} \land m \in \text{dom(sender)} \]
\[ \land p_1 \neq \text{sender(m)} \land \text{VTP}(p_1(p)) \geq \text{VTM}(m(p)) \Rightarrow (p_1 \rightarrow m) \in \text{receive} ) \]

\[ \land \forall m, p. (p \in \text{PROCESS} \land m \in \text{MESSAGE} \land m \in \text{dom(sender)} \]
\[ \Rightarrow \text{VTM}(m(p)) \leq \text{VTP}(p)(p) ) \]

\[ \land \forall p_1, p_2. (p_1 \in \text{PROCESS} \land p_2 \in \text{PROCESS} \land p_1 \neq p_2 \Rightarrow \text{VTP}(p_1(p_2)) \leq \text{VTP}(p_2)(p_2)) \]
Conclusions

- We outlined how an abstract causal order is correctly implemented through vector clocks.

- Ordered delivery of messages may provide enough information needed at the time of recovery from failures.

- Adequacy of Event B to provide a complete framework for developing mathematical models of distributed algorithms.

- Illustration of use of Event B for rigorous description of problem, gradual refinement to more concrete specifications and verification of solution for correctness.