INFO8002

Large-Scale Data Systems

Exercise Session #1

Academic year 2021-2022

LIÈGE université

CONTACT

- Ben Mariem Sami
- Office 1.14 - B37 Institut Mathématique
- sami.benmariem@uliege.be

- https://github.com/glouppe/info8002-large-scale-data-systems

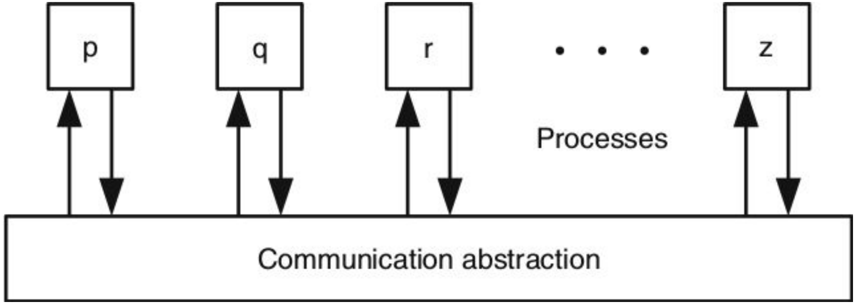
ORGANISATION

- Evaluation:
  - Reading assignment → 10% of the final mark
  - Project 1 → 40% of the final mark
  - Oral Exam → 50% of the final mark

Reminder

REMINDER :

Asynchronous Event-based Composition Model



- A **distributed algorithm** is a distributed collection Π = {p,q,r,...} of **N** processes implemented by finite state automata.4

- Event-based **component** or **module** model:
    - Each program consists of a set of modules.
    - Modules interact via **events**.

REMINDER :

Asynchronous Event-based Composition Model

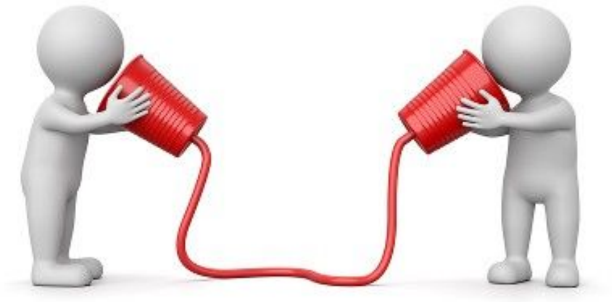
- **Asynchronous events** represent communication or control flow between components:
    - Each component is constructed as a state-machine whose transitions are triggered by the reception of events.
    - Events carry information (sender, message, etc)
- Code for each **component** looks like this:

upon event <Module1, Event1 | att^1, att^2, ...> **do**
    ...
    trigger <Module2, Event2 | att^1, att^2, ...>; //Trigger some events

PROBLEM 1

Peer-To-Peer Messaging System

You are responsible for creating a **peer-to-peer** messaging system.



Specify a link abstraction module for message delivery between peers, and provide a pseudo-implementation using sequence numbers.

Peer-To-Peer Messaging System

Problems:

1. **How can we ensure that messages are eventually delivered?**

Peer-To-Peer Messaging System

Problems:

1. **How can we ensure that messages are eventually delivered?**

   Property n°1: **Reliable Delivery**:

   "If a correct process ***p*** *sends a message* ***m*** *to a correct process* ***q****, then* ***q*** *eventually delivers* ***m****."*

Peer-To-Peer Messaging System

Problems:

1. **How can we ensure that messages are eventually delivered?**

   Property n°1: **Reliable Delivery**:

   "If a correct process ***p*** *sends a message* ***m*** *to a correct process* ***q****, then* ***q*** *eventually delivers* ***m****."*

2. **How can we ensure that no messages are delivered more than once ?**

Peer-To-Peer Messaging System

Problems:

1. **How can we ensure that messages are eventually delivered?**

   Property n°1: **Reliable Delivery**:

   *"If a correct process **p** sends a message **m** to a correct process **q**, then **q** eventually delivers **m**."*

2. **How can we ensure that no messages are delivered more than once ?**

   Property n°2: **No Duplication**:

   "No message is delivered by a process more than once."

Peer-To-Peer Messaging System

Problems:

1. **How can we ensure that messages are eventually delivered?**

   Property n°1: **Reliable Delivery**:

   "If a correct process ***p*** *sends a message* ***m*** *to a correct process* ***q****, then* ***q*** *eventually delivers* ***m****."*

2. **How can we ensure that no messages are delivered more than once ?**

   Property n°2: **No Duplication**:

   "No message is delivered by a process more than once."

3. **How can we ensure that messages that has been delivered has been sent by some other process?**

Peer-To-Peer Messaging System

Problems:

1. **How can we ensure that messages are eventually delivered?**

   Property n°1: **Reliable Delivery**:

   "If a correct process ***p*** *sends a message* ***m*** *to a correct process* ***q****, then* ***q*** *eventually delivers* ***m****."*

2. **How can we ensure that no messages are delivered more than once ?**

   Property n°2: **No Duplication**:

   "No message is delivered by a process more than once."

3. **How can we ensure that messages that has been delivered has been sent by some other process?**

   Property n°3: **No Creation**:

   "If some process ***q*** *delivers a message* ***m*** *with sender* ***p****, then* ***m*** *was previously sent to* ***q*** *by process* ***p****."*

Peer-To-Peer Messaging System

Problems:

4. **How can we ensure that messages are delivered <u>in order</u>?**

Peer-To-Peer Messaging System

Problems:

4. **How can we ensure that messages are delivered in order?**

   Property n°4: **FIFO delivery**:

   *"If some process **p** sends message **m1** before it sends message **m2**, then no correct process delivers **m2** unless it has already delivered **m1**."*

Peer-To-Peer Messaging System

Module Specification:

Module 1: Interface and properties of peer-to-peer messaging links

Module:

Name: *MessagingLinks*, **instance** *ml***.**

Events:

Request: <ml, Send | q, m> : Requests to send message **m** to process **q**.

Indication: <ml, Deliver | p, m> : Delivers message **m** sent by process **p**.

Properties:

ML1: *Reliable delivery.*

ML2: *No duplication.*

ML3: *No creation.*

ML4: *FIFO delivery.*

(ML1–ML3:) *"Perfect point-to-point links"* Module

Peer-To-Peer Messaging System

Implementation

Algorithm 1: Sequence Number

Implements:

MessagingLinks, **instance** *ml*.

Uses:

PerfectPointToPointLinks, **instance** *pl*.

upon event <ml, *Init*> **do**

???

upon event <ml, *Send* | q, m> **do**

???

upon event <pl, Deliver | p, (m, sn)> **do**

???

Peer-To-Peer Messaging System

Implementation

```
Algorithm 1: Sequence Number
    Implements:
        MessagingLinks, instance ml.
    Uses:
        PerfectPointToPointLinks, instance pl.

    upon event <ml, Init> do
        forall p ∈Π do
            lsn[p] := 0;
            next[p] := 1;

    upon event <ml, Send | q, m> do
        lsn[q] := lsn[q] + 1;
        trigger <pl, Send | q, (m, lsn[q])>;

    upon event <pl, Deliver | p, (m, sn)> do
        ???
```

Peer-To-Peer Messaging System

Implementation

Algorithm 1: Sequence Number

```
Implements:
    MessagingLinks, instance ml.
Uses:
    PerfectPointToPointLinks, instance pl.

upon event <ml, Init> do
    forall p ∈ Π do
        lsn[p] := 0;
        next[p] := 1;

upon event <ml, Send | q, m> do
    lsn[q] := lsn[q] + 1;
    trigger <pl, Send | q, (m, lsn[q])>;

upon event <pl, Deliver | p, (m, sn)> do
    pending := pending ∪ {(p, m, sn)};
    while exists (q, n, sn') ∈ pending such
that sn' = next[q] do
        next[q] := next[q] + 1;
        pending := pending \ {(q, n, sn')};
        trigger <ml, Deliver | q, n>
```

PROBLEM 2

Peer-To-Peer Messaging System

You are responsible for creating a **peer-to-peer** messaging system with messaging room.

Specify a broadcast abstraction module for message delivery to all peers in the same messaging room as the sender.

Peer-To-Peer Messaging System

Problems:

1. **How can we ensure that messages are eventually delivered?**

Peer-To-Peer Messaging System

Problems:

1. **How can we ensure that messages are eventually delivered?**

   Property n°1: **Validity**:

   *"If a correct process **p** broadcasts a message **m**, then every correct process eventually delivers **m**."*

Peer-To-Peer Messaging System

Problems:

1. **How can we ensure that messages are eventually delivered?**

   Property n°1: **Validity**:

   *"If a correct process **p** broadcasts a message **m**, then every correct process eventually delivers **m**."*

2. **How can we ensure that no messages are delivered more than once ?**

Peer-To-Peer Messaging System

Problems:

1. **How can we ensure that messages are eventually delivered?**

   Property n°1: **Validity**:

   *"If a correct process **p** broadcasts a message **m**, then every correct process eventually delivers **m**."*

2. **How can we ensure that no messages are delivered more than once ?**

   Property n°2: **No Duplication**:

   "No message is delivered by a process more than once."

3. **How can we ensure that messages that has been delivered has been sent by some other process?**

Peer-To-Peer Messaging System

Problems:

1. **How can we ensure that messages are eventually delivered?**

   Property n°1: **Validity**:

   *"If a correct process **p** broadcasts a message **m**, then every correct process eventually delivers **m**."*

2. **How can we ensure that no messages are delivered more than once ?**

   Property n°2: **No Duplication**:

   "No message is delivered by a process more than once."

3. **How can we ensure that messages that has been delivered has been sent by some other process?**

   Property n°2: **No Creation**:

   *"If a process delivers a message **m** with sender **s**, then **m** was previously broadcast by process **s**."""*

Peer–To–Peer Messaging System

Problems:

4. **How can we ensure that if sender crashes, all or none of the correct node deliver the message?**

Peer-To-Peer Messaging System

Problems:

4. **How can we ensure that if sender crashes, all or none of the correct node deliver the message?**
   Property n°4: **Agreement**::
   *"If a message **m** is delivered by some correct process, then **m** is eventually delivered by every correct process."*

5. **How can we ensure that messages are delivered in order?**

Peer-To-Peer Messaging System

Problems:

4. **How can we ensure that if sender crashes, all or none of the correct node deliver the message?**

   Property n°4: **Agreement**::

   *"If a message **m** is delivered by some correct process, then **m** is eventually delivered by every correct process."*

5. **How can we ensure that messages are delivered in order?**

   Property n°4: **FIFO delivery**:

   *"If some process **p** broadcast message **m1** before it broadcast message **m2**, then no correct process delivers **m2** unless it has already delivered **m1**."*

Peer-To-Peer Messaging System

Module Specification:

Module 2: Interface and properties of peer-to-peer messaging broadcast

Module:

Name: FIFO*MessagingBroadcast*, **instance** *fmb*.

Events:

Request: <*fmb*, Broadcast | m> : Requests to broadcast message **m**.

Indication: <*fmb*, Deliver | p, m> : Delivers message **m** broadcast by process **p**.

Properties:

FMB1: *Validity.*

FMB2: *No duplication.*

FMB3: *No creation.*

FMB4: *Agreement.*

FMB5: *FIFO delivery.*

(FMB1–FMB4: *"Reliable Broadcast"* Module)

Peer-To-Peer Messaging System

Implementation

Algorithm 2: Sequence Number Broadcast

Implements:

FIFO*MessagingBroadcast*, **instance** *fmb*.

Uses:

ReliableBroadcast, **instance** *rb*.

upon event <*fmb*, *Init*> **do**

???

upon event <rb, Deliver | p, [DATA,s, m,sn]> **do**

???

upon event <*fmb*, Broadcast | m> **do**

???

Peer-To-Peer Messaging System

Implementation

Algorithm 2: Sequence Number Broadcast

Implements:

FIFO*MessagingBroadcast*, **instance** *fmb*.

Uses:

ReliableBroadcast, **instance** *rb*.

upon event <*fmb*, *Init*> **do**

lsn := 0;

pending := ∅;

next := $[1]^N$;

upon event <*fmb*, Broadcast | m> **do**

lsn := lsn + 1 ;

trigger <rb, Broadcast | [DATA, self, m, lsn] >;

upon event <rb, Deliver | p, [DATA,s, m,sn]> **do**

pending := *pending* ∪ {(s, m, sn)};

while exists (s, m', sn') ∈ *pending* **such that** *sn'*= *next*[s] **do**

next[s] := *next*[s] + 1 ;

pending := *pending* \{(s, m', sn')};

trigger <frb, Deliver | s,m'>;

Peer-To-Peer Messaging System

Problems:

4. **How can we ensure that if sender crashes, all or none of the correct node deliver the message?**

   Property n°4: **Agreement**::

   *"If a message **m** is delivered by some correct process, then **m** is eventually delivered by every correct process."*

5. **How can we ensure that messages are delivered in order?**

   Property n°4: **FIFO delivery**:

   *"If some process **p** broadcast message **m1** before it broadcast message **m2**, then no correct process delivers **m2** unless*

PROBLEM:

Sender	Message
Mr. X	Where is the lecture?
Mr. X	Thank you!
Mrs. Y	R3.

Peer-To-Peer Messaging System

Problems:

4. **How can we ensure that if sender crashes, all or none of the correct node deliver the message?**
   Property n°4: **Agreement**::
   *"If a message **m** is delivered by some correct process, then **m** is eventually delivered by every correct process."*

5. **How can we ensure that messages are delivered in order?**
   Property n°4: **Causal delivery**:
   *"For any message **m1** that potentially caused a message **m2**, i.e., **m1** →**m2**, no process delivers **m2** unless it has already delivered **m1**."*

Peer-To-Peer Messaging System

Module Specification:

Module 2: Interface and properties of peer-to-peer messaging broadcast

Module:

Name: Causal*MessagingBroadcast*, **instance** *cmb*.

Events:

Request: <*cmb*, Broadcast | m> : Requests to broadcast message **m**.

Indication: <*cmb*, Deliver | p, m> : Delivers message **m** broadcast by process **p**.

Properties:

FMB1: *Validity.*
FMB2: *No duplication.*
FMB3: *No creation.*
FMB4: *Agreement.*
FMB5: *Causal delivery.*

"Causal Order Reliable Broadcast" Module
(cfr. lecture 3)

Peer-To-Peer Messaging System

Problems:

4. **How can we ensure that if sender crashes, <u>all or none</u> of the correct node deliver the message?**

   <u>Property n°4: **Agreement**:</u>:

   *"If a message **m** is delivered by some correct process, then **m** is eventually delivered by every correct process."*

5. **How can we ensure that messages are delivered <u>in order</u>?**

   <u>Property n°4: **Causal delivery**:</u>

   *"For any message **m1** that* … *→**m2**, no process delivers **m2** unless it has already deliv*…

PROBLEM:

Sender	Message
Mr. X	Where is the lecture?
Mrs Y	R3.
Mrs. Z	Where is the lecture?
Mr. X	Thank you!

Peer-To-Peer Messaging System

Problems:

4. **How can we ensure that messages are delivered in order?**

Property n°4: **Total Order delivery**:

*"If correct processes **pi** and **pj** both deliver messages **m1** and **m2**, then **pi** delivers **m1** before **m2 IFF** process **pj** delivers **m2** before **m1**."*

Peer-To-Peer Messaging System

Problems:

4. **How can we ensure that messages are delivered in order?**

Property n°4: **Total Order delivery**:

*"If correct processes **pi** and **pj** both deliver messages **m1** and **m2**, then **pi** delivers **m1** before **m2 IFF** process **pj** delivers **m2** before **m1**."*

- *In an asynchronous system?*
- *In a partially asynchronous system?*
- *In a synchronous system?*

HOMEWORK !