Selective choice ‘feathering’ with XCHANs

Øyvind Teig
CPA-2013 at Edinburgh Napier University, Scotland

Øyvind Teig

Autronica Fire and Security
Trondheim, Norway

http://www.teigfam.net/oyvind/pub/pub_details.html#FEATHERING
http://wotug.org/paperdb/

Edinburgh, 27Aug2013
«Feathering»

• Semantics of a verb *to uninterest*

• Avoiding the uninteresting

• Taking *uninterestedness* seriously
Background of the XCHAN paper (2012)

- From discussions at Autronica
- Not implemented
- Goal for me was to try to merge asynchronous and synchronous "camps"..
- ..to arrive at a common methodology
- To make it "easier" to comply to SIL (Safety Integrity Level) approving according to IEC 61508 standard for safety critical systems
- Assumed implementation loosely based on implemented ideas with EGGTIMER and REPTIMER. ([9] CPA-2009 paper)
\[ \text{XCHAN} = \text{x-channel} + \text{CHAN} \]

Figure 1. \text{XCHAN} is \text{CHAN} plus \text{x-channel}
This paper uses

«classic» solution (from 2012 XCHAN paper)

as opposed to occam-pi model of XCHAN(*)

«preconfirmed»

(*) Peter H. Welch. An occam Model of XCHANs, 2013.
https://www.cs.kent.ac.uk/research/groups/plas/wiki/An_occam_Model_of_XCHANs
XCHAN (...) OF BYTE my_xchan:

Sender is notified as to its success or "failure"
XCHAN (...) OF BYTE my_xchan:

Sender is notified as to its success on return of send:
- data moved to buffer
- data moved to receiver
XCHAN (...) OF BYTE my_xchan:

Sender is notified as to its "failure" on return of send:
- buffer full
- receiver not present
XCHAN (...) OF BYTE my_xchan:

Sender is notified as to its "failure" on return of send:
- buffer full
- receiver not present

It always returns!
If "failed" to send on XCHAN:
If "failed" to send on XCHAN:

"Not sent" is no fault!
If "failed" to send on XCHAN:

"Not sent" is no fault!

But a contract to send later
If not sent on XCHAN:

- **listen to x-channel** (in an ALT or select)
- **resend** old or fresher value when it arrives
- this send will always succeed
If not sent:

- listen to x-channel (in an ALT or select)
- resend old or fresher value when it arrives
- this send will always succeed
If not sent:

- listen to x-channel (in an ALT or select)
- resend old or fresher value when it arrives
- this send will always succeed

This contract (design pattern) between sender and receiver must be adhered to
Feathering
Ripping a term

- “Turning an oar parallel to the water between pulls”
- But we can hear the oar whip the top of the small waves on its way saying “was there, but not interested”
- So we take the step to name barely touching the small waves as feathering
- And give feathering a new meaning
«Concurrent programs wait faster»

- Tony Hoare’s lecture from 2003
- Not waiting for a certain bus, but for correct destination..
- ..makes us «wait faster», but..
- XCHAN as a vehicle for a secondary problem not mentioned in Hoare’s lecture:
Also for non-interesting buses!

- What happens after the first possible bus has arrived is not treated here.

- What happens with uninteresting buses while waiting, we have specifically said is not of interest - but..

- ..why do we still have to relate to these bus arrivals afterwards?
I said *not-interesting* buses!

- There is no way to avoid having to flush these messages!
- But we could have avoided sending them!
You’re sitting on the first relevant bus, but its conductor requires you to pay for all the buses that stopped while you waited!

- Sending unnecessarily is as bad as paying unnecessarily.
- This is state of the art, also for occam!
- Simply because a blocked sender has only one way to unblock: to get rid of its message.
Figure 2. **XCHAN** (array of 3) and feathering, with only bus #1 as possible to ride
Feathering semantics (1/10)

1. Feathering semantics inherits XCHAN semantics
   
a. Output and input constructs limitations (next page)
   
b. This may not include buffered XCHAN: usability analysis needed
Feathering semantics

Where to use it

2. **Receiver** end of XCHAN in ALT, not single channel input.

**Sending** end single channel output, not part of an ALT with output guards (XCHAN almost eq. to an output guard)
Feathering semantics
User control

3. Specified with a parameter in the XCHAN send call (not in examples here)
Feathering semantics

Already not interested

4. Feathered status call reply to a sender that is trying to send when a receiver is in an ALT and the requested channel has been tagged by the receiver as not-interesting (i.e. its pre-condition is FALSE)
5. X-feathered status messaged response is sent to a sender on x-channel if it has been trying to send but got await_commit reply; when the receiver enters an ALT and the requested channel is being tagged as not-interesting

a. Only if the ALT blocks - i.e. it is not immediately taken by another guard (channel, timeout or SKIP)

b. None of the receivers will block indefinitely, commitment to listen on x-channel
Feathering semantics

Usage rule

6. Whenever a sender knows that a channel is feathered it shall obey the rule not to resend before an x-unfeathered message has been received on x-channel
Feathering semantics

Perhaps interested next time, so..

7. The $x$-unfeathered status is delivered to a feathered $x$-channel when the ALT is later on taken (by another guard) and ‘torn down’, in the same synchronous scheme as described above (5.a-b)
Feathering semantics

Only tell if it is in scope

8. The x-channel will only carry an x-unfeathered after a feathered situation
Feathering semantics

Standard CHAN semantics if sent on first trial

9. The x-channel will only carry x-feathered or x-committed after an await_commit status return on the initial sending call
Feathering semantics (last)

10. A receiver could possibly do a system call to learn if a message in fact did get rejected. This information could alternatively be delivered on an "n-channel" that could be “parallel” with the XCHAN’s input on the receiver side. This probably is a complicating matter since type of channel is transparent on the receiver side. We will not discuss this here.
Figure 3. Two mind map scenarios that show message avoidance
while (TRUE) {
  ALT();
  ALT_SIGNAL_CHAN_IN (XCHAN_READY); // data-less
  ALT_CHAN_IN (CHAN_DATA_IN, Value);
  ALT_END(); // Delivers ThisChannelId:

  switch (ThisChannelId) {
    case XCHAN_READY: {
      // sending will succeed
      CP->Sent_Out = CHAN_OUT (XCHAN_DATA_OUT, Value);
      break;
    }
    case CHAN_DATA_IN: {
      if (!CP->Sent_Out) {
        ... handle overflow (decide what value(s) to discard)
      } else {
        // sending may succeed:
        CP->Sent_Out = CHAN_OUT (XCHAN_DATA_OUT, Value);
      }
      break;
    }
    _DEFAULT_EXIT_VAL (ThisChannelId)
  }
}
An XCHAN feathering solution (code)

```c
01 CP->Tag = READY; // READY,SUCCESS,AWAIT_READY,FEATHERED
02 while (TRUE) {
03     PRIALT();
04     ALT_CHAN_IN (X_CHANNEL,X_Tag); // X_COMMITTED,
05           // X_FEATHERED,X_UNFEATHERED
06     ALT_CHAN_IN (CHAN_DATA_IN,Value);
07? ALT_END(); // Delivers ThisChannelId
08
09     switch (ThisChannelId) {
10         case X_CHANNEL: { // After CHAN_OUT ret AWAİT READY or FEATHERED
11             if (X_Tag == X_FEATHERED) {
12                 ... handle not interested
13             CP->Tag = FEATHERED; // Stop
14         } else if (X_Tag == X_COMMITTED){
15! CHAN_OUT (XCHAN_DATA_OUT,Value,NIL); // Will succeed
16             CP->Tag = READY; // Finished
17         } else { // == X_UNFEATHERED
18             CP->Tag = READY; // Finished
19         }
20     } break;
21     case CHAN_DATA_IN: {
22         if (((CP->Tag == AWAİT READY) or (CP->Tag == FEATHERED)) { // CP->Tag = READY
23             ... handle overflow (decide what value(s) to discard)
24         } else { // CP->Tag = READY
25             CP->Tag = CHAN_OUT (XCHAN_DATA_OUT,Value,ALLOW_FEATHERING);
26             if (CP->Tag == SUCCESS) {
27                 CP->Tag = READY; // Finished
28         } else if (CP->Tag == FEATHERED) {
29             ... handle not interested
30         } else { // CP->Tag == AWAİT READY
31         }
32     } break;
33     }
34 }
35 }
```

Listing 1. Overflow and ‘feathered’ handling on an XCHAN (ANSI C and macros)
Listing 1. Overflow and ‘feathered’ handling on an XCHAN (ANSI C and macros)
occam semantic non-equivalence

However, the ALT in line 12 will never take part in any feathering, neither will the two SEQ blocks starting at lines 18 and 22.

Listing 2 - Feathering loss of semantic equivalence (occam)
Asymmetry aspect 1

• Receiver defines when it is not interested (time window)

• But sender does not know about this (or anything at all) before it tries to send
Asymmetry aspect 2

- Sender gets to know that something had been deemed noninteresting by the receiver
- But receiver does not know that something consequently has not been sent
Pattern extends ALT up to a certain level only

- However, extra «symmetrifying» messaging for this will fast take us into application level publish-subscribe pattern

- This is the price for keeping a «clean» ALT
Uninterestingness is treated with application level receiver’s ALT transparently

Cycles saved should be more than cycles taken

This will depend on message length
Abstraction

• Not having to send and not having to treat not-interesting messages is an «abstraction gain»

• «Cognitive message clutter» avoided

• Increases «non-determinism»
Safe

- XCHAN with feathering is a safe concept:
- Overflow and dropping of uninteresting messages are both handled at application level
- No overflow like malloc heap overflow, which causes restart
Selective choice ‘feathering’ with XCHANs

Questions?

About the two pictures in the last slide
Pictures

Front picture is a base of a structure in Porto Antico in Genova, Italy. It holds a large tent, an elevator basket etc.

The last picture is from Museum Villa Croce Contemporary in Genova, where we discovered a student uninteresting a text for a previous exhibition

Both © Øyvind Teig, 2013

Thank you!