Extensions to the Concurrent Communications Library

Kenneth Skovhede
Niels Bohr Institute
University of Copenhagen
Extensions to the Concurrent Communications Library

Kenneth Skovhede
Niels Bohr Institute
University of Copenhagen

CPA 2016 - Copenhagen
Background

Overview
A C-like library for C and other C-derived languages

- Chaotic yet systematic
- Syntax faithful to C
- Lookahead for language syntax
- Event callback hooks

Features
- Only one channel type
  - generates full type-safety
- Named and synonym channels
- Sequence guarantees through named fans
- SafeView commit used for guaranteed
  shared prevention through of data races
- Allocation with input or output local timeouts
- Segments exposed through immediate invariants
- Hardcoded channel rate, features disabled
Overview

A CSP inspired library in/for C# (and other CIL languages)

- Utilizes await/async
- Builds FSMs from sequential code
- Low memory overhead pr. process
- Fewer context switches

while(true)
    await out.WriteAsync(
        await in.ReadAsync());
Features

- Only one channel type, generics for type-safety
- Named and anonymous channels
- Sequence guarantee through request lists
- Alternation with priority, random, and fair select
- Two-phase commit used to guarantee atomic selection external to the channel
- Alternation with input or output and timeouts
- Skip guards supported through immediate timeouts
- Minimalistic channel core, features outside
Productivity enhancements
Mixing read and write requests

```csharp
var cur = 0, prev = 0;
var updatechan = ChannelManager.GetChannel<string>("update");
var curchan = ChannelManager.GetChannel<int>("cur");
var prevchan = ChannelManager.GetChannel<int>("prev");

while(true) {
    var res = await MultiChannelAccess.ReadOrWriteAnyAsync(
        MultisetRequest.Read(updatechan),
        MultisetRequest.Write(cur, curchan),
        MultisetRequest.Write(prev, prevchan),
    );

    if (res.IsRead) {
        prev = cur;
        cur = int.Parse((string)res.Value);
    }
}
```
Overflowing a channel

```csharp
var warn = ChannelManager.GetChannel<int>("warn");
var report = ChannelManager.GetChannel<int>("report");

while(true) {
    var current = ReadHardwareSensor();
    if (current > 10)
    {
        warn.Write(current);
        report.Write(current);
    }
}

var warn = ChannelManager.GetChannel<int>("warn");
var report = ChannelManager.GetChannel<int>("report",
                                      maxPendingWriters: 1,
                                      pendingWritersOverflowStrategy:
                                          QueueOverflowStrategy.FIFO
                                      );

while(true) {
    var current = ReadHardwareSensor();
    if (current > 10)
    {
        warn.Write(current);
        report.WriteNoWait(current);
    }
}
```

- Enables Actor-like paradigm
- The WriteAsync() method returns a value that can be checked for failure
Channel scopes

```csharp
var external = ChannelManager.GetChannel<int>("out");
using(new IsolatedScope())
    await Task.WhenAll(
        Task.Run(async () => { // Process 1
            var internal = 
                ChannelManager.GetChannel<int>("out");

            while(true)
                await internal.WriteAsync(42);
        })),

        Task.Run(async () => { // Process 2
            var internal = 
                ChannelManager.GetChannel<int>("out");

            while(true)
                await external.WriteAsync(
                    await internal.ReadAsync());
        }));
```
Wiring channels automatically

**Protonein Network in PyCSP**

```
// PyCSP

Producer(producer_read, collector_write, done_write)
Process(worker_read, feeder_read, collector_write, done_write)
Process(worker_read, feeder_read, collector_write, done_write)
Process(worker_read, feeder_read, collector_write, done_write)
Process(worker_read, feeder_read, collector_write, done_write)
Process(worker_read, feeder_read, collector_write, done_write)
Process(distributor, collector_read)
```

**Protonein Network in CoCoL**

```
var feeder = ChannelManager.getChannelInt();
var collector = ChannelManager.getChannelInt();
var done = ChannelManager.getChannelBool();

Task whenAll:
  Producer(protein, map, place, feeder.AWrite());
  Worker(feeders, AsRead(), collector.AWrite(), done.AWrite());
  Worker(feeders, AsRead(), collector.AWrite(), done.AWrite());
  Worker(feeders, AsRead(), collector.AWrite(), done.AWrite());
  Worker(feeders, AsRead(), collector.AWrite(), done.AWrite());
  Worker(feeders, AsRead(), collector.AWrite(), done.AWrite());
  Worker(feeders, AsRead(), collector.AWrite(), done.AWrite());
  Barrier(done.ARead());
  collector.AWrite();
  done.AWrite();
```

**Protonein Network with Wiring**

```
public class Worker : ProcessHelper {
  [ChannelName("feeder")]
  private IReadChannel<int> feeder;

  [ChannelName("collector")]
  private IWriteChannel<int> collector;

  [ChannelName("done")]
  private IWriteChannel<bool> done;

  ... code omitted ...
}

Task whenAll:
  new Producer(protein, map, place);
  new Worker(), new Worker(), new Worker(),
  new Worker(), new Worker(),
  new Barrier();
  new Consumer();
```
Prototein Network in PyCSP

feeder = One2AnyChannel()
collector = Any2OneChannel()
done = Any2OneChannel()

Parallel(
    Process(producer, protein, map, place, feeder.write),
    Process(worker, feeder.read, collector.write, done.write),
    Process(worker, feeder.read, collector.write, done.write),
    Process(worker, feeder.read, collector.write, done.write),
    Process(worker, feeder.read, collector.write, done.write),
    Process(barrier, 5, done.read, collector.write),
    Process(consumer, collector.read))
Prototein Network in CoCoL

```csharp
var feeder = ChannelManager.GetChannel<int>();
var collector = ChannelManager.GetChannel<int>();
var done = ChannelManager.GetChannel<bool>();

Task.WhenAll(
    Producer(protein, map, place, feeder.AsWrite()),
    Worker(feeder.AsRead(), collector.AsWrite(), done.AsWrite()),
    Worker(feeder.AsRead(), collector.AsWrite(), done.AsWrite()),
    Worker(feeder.AsRead(), collector.AsWrite(), done.AsWrite()),
    Worker(feeder.AsRead(), collector.AsWrite(), done.AsWrite()),
    Worker(feeder.AsRead(), collector.AsWrite(), done.AsWrite()),
    Barrier(5, done.AsRead(), collector.AsWrite()),
    Consumer(collector.AsRead()));
```
public class Worker : ProcessHelper {
    [ChannelName("feeder")]
    private IReadChannel<int> feeder;

    [ChannelName("collector")]
    private IWriteChannel<int> collector;

    [ChannelName("done")]
    private IWriteChannel<bool> done;

    ... code omitted ...
}

Task.WhenAll(
    new Producer(protein, map, place),
    new Worker(), new Worker(), new Worker(),
    new Worker(), new Worker(),
    new Barrier(5),
    new Consumer());
Leave and Join

Solution:
- Readers and writers "join" the channel
- Keep a counter for number of readers and number of writers
- When poisoned, "leave" the channel
- When either counter reaches zero poison the channel

*Same as in PyCSP Revisited, but differs in handling poison/retire*

```java
private static Task Worker() {
    return AutoTaskExtensions.RunTask(
        new [] {
            feed = channelMarker.ForReading("feeder"),
            coll = channelMarker.ForWriting("collector"),
        },
        async { self } => {
            while (true) {
                var item = await self.feed.ReadAsync();
                ... code omitted ...
                await self.coll.WriteAsync(result);
            }
        });
}
```

Adding Lambda Closures
Solution:

- Readers and writers "join" the channel
- Keep a counter for number of readers and number of writers
- When poisoned, "leave" the channel
- When either counter reaches zero poison the channel

*Same as in PyCSP Revisited, but differs in handling poison/retire*
private static Task Worker () {
    return AutomationExtensions.RunTask(
        new {
            feed = ChannelMarker.ForRead<int>("feeder"),
            coll = ChannelMarker.ForWrite<int>("collector")
        },
        async (self) => {
            while (true) {
                var data = await self.feed.ReadAsync();
                ... code omitted ...
                await self.coll.WriteAsync(result);
            }
        }
    );
}

Task.WhenAll(
    Producer(protein, map, place),
    Worker(), Worker(), Worker(), Worker(), Worker(), Worker(),
    Consumer());
Network support

Design ideas

-过多的冗余信息可能降低系统的性能
-需要识别和过滤不必要的信息
-应考虑数据的组织方式和传输路径
-应优化数据的编码和解码过程
-可能需要使用更高效的传输协议
-应确保传输的数据具有安全性和可靠性

Diagram:

- 8 bytes total length
- N bytes header data
- 8 bytes payload length
- payload data

Diagram:

Diagram:

Diagram:
Design ideas

Drop-in replacement channel, user should not take special care

3-tier setup:
- Name-/discovery-servers
- Channel hosting servers
- Clients

* Currently has name+hosting in one

Channel hosting server should only relay to/from network to a normal hosted channel: no special logic for a network channel

Timeout handled on hosting server

Two-phase commit support, enables ALT'ing with any type channel-mix
2 bytes header length

8 bytes total length

N bytes header data

8 bytes payload length

payload data
Fighting latency with buffers....

Experimental feature:
use "windowing" writes and reads to hide latency

- Writes succeed locally
  - And emits WriteRequest
- When window is full
  - Wait until WriteResponse

- First read emits N ReadRequests
  - Then waits for ReadResponse
- Next read emits 1 ReadRequest
  - And wait for oldest ReadResponse

* Breaks CSP guarantees
* A wrapper on top of a channel
Experiments

CommsTime

StressedAlt

Mandelbrot
CommsTime

Tick channel using loopback network

All channels using loopback network

Tick channel using remote channel server

All channels using remote channel server
Mandelbrot using remote channel server

Microseconds per pixel

Buffer size

- 25x25
- 50x50
- 100x100
StressedAlt

Alternation using loopback network

Alternation using remote channel server
Future work

Error handling, transaction logs

Skeleton methods

Channel mobility
Thanks!

Source code with benchmarks and examples on GitHub: https://github.com/kenkendk/cocol
Extensions to the Concurrent Communications Library

Kenneth Skovhede
Niels Bohr Institute
University of Copenhagen

CPA 2016 - Copenhagen