Communicating Complex Systems

transforming formal models of complex systems into executable code
implementing complex functionality in and for occam-$\pi$

Joint work with Peter Welch (University of Kent)
and Fiona Polack (University of York)
Contents

- Motivation
- A quick introduction to occam-π
- A simplified blood-clotting model
- Transformation into occam-π
- Multi-way synchronisation in occam-π
- Further developments
- Conclusions
Motivation
We want to build complex systems, interested in emergent behaviour

- modelling biological systems — in this case, blood clotting (haemostatis)
- discovering the parameters of systems needed to do this: can we do it with 10 ‘processes’? a thousand? a million?
  what concurrency abstractions are needed?
- systems built from independent, interacting agents, operating within some environment — an approximation of the real-world
We want to build complex systems, interested in emergent behaviour

- modelling biological systems — in this case, blood clotting (haemostatis)
- discovering the parameters of systems needed to do this: can we do it with 10 ‘processes’? a thousand? a million?
- what concurrency abstractions are needed?
- systems built from independent, interacting agents, operating within some environment — an approximation of the real-world

Need to be sure that our agents are safe in the implementation

- making sure that when we put hundreds of thousands together, the whole thing won’t deadlock, and that the results we observe are trustworthy
- use process calculi such as Circus [Woodcock et al.], CSP [Hoare] and the $\pi$-calculus [Milner] to formally reason about such systems
Motivation
Motivation

Using *occam*-π for the implementation

- CSP related semantics, with ideas of mobility from the π-calculus
- ultra-lightweight concurrency overheads — can handle up to a million simple processes on a modern desktop PC
- clean mapping from formal specification to implementation
Motivation

Using **occam-π** for the implementation

- CSP related semantics, with ideas of mobility from the \( \pi \)-calculus
- ultra-lightweight concurrency overheads — can handle up to a million simple processes on a modern desktop PC
- clean mapping from formal specification to implementation

Driving forward the development of **occam-π**

- our models make heavy use of **multi-way synchronisations**
- processes (agents) may make and withdraw offers of their own accord
- require an efficient and safe implementation for this, which previously did not exist in **occam-π**
A Quick Introduction to occam-$\pi$
A Quick Introduction to occam-\(\pi\)

- Systems are built from layered **networks** of communicating **processes**
  - linkage with single **channels** (synchronised, unbuffered point-to-point communication) and **bundles** of channels
  - extended to handle **shared** channels (communication still between two processes only) — also **barriers**, but only committed synchronisations
  - dynamic process creation, and more ...
A Quick Introduction to occam-$\pi$

- Systems are built from layered **networks** of communicating **processes**
  - linkage with single **channels** (synchronised, unbuffered point-to-point communication) and **bundles** of channels
  - extended to handle **shared** channels (communication still between two processes only) — also **barriers**, but only committed synchronisations
  - dynamic process creation, and more ...
A Quick Introduction to occam-$\pi$

Systems are built from layered **networks** of communicating **processes**

- linkage with single **channels** (synchronised, unbuffered point-to-point communication) and **bundles** of channels
- extended to handle **shared** channels (communication still between two processes only) — also **barriers**, but only committed synchronisations
- dynamic process creation, and more ...

![Diagram of a simple system with interconnections labeled a, b, c, and δ.](image-url)
A Quick Introduction to occam-π

- Systems are built from layered **networks** of communicating **processes**
  - linkage with single **channels** (synchronised, unbuffered point-to-point communication) and **bundles** of channels
  - extended to handle **shared** channels (communication still between two processes only) — also **barriers**, but only committed synchronisations
  - dynamic process creation, and more ...

Fred Barnes, August 2006
A Quick Introduction to occam-$\pi$

- Systems are built from layered **networks** of communicating **processes**
  - linkage with single **channels** (synchronised, unbuffered point-to-point communication) and **bundles** of channels
  - extended to handle **shared** channels (communication still between two processes only) — also **barriers**, but only committed synchronisations
  - dynamic process creation, and more ...

---

Fred Barnes, August 2006
Systems are built from layered **networks** of communicating **processes**

- linkage with single **channels** (synchronised, unbuffered point-to-point communication) and **bundles** of channels
- extended to handle **shared** channels (communication still between two processes only) — also **barriers**, but only committed synchronisations
- dynamic process creation, and more ...

---

![Diagram](image.png)
A Quick Introduction to occam-$\pi$
A Quick Introduction to occam-π

- occam-π has been used successfully to create systems which could be described as ‘complex’
  - operating-system components (the RMoX system)
  - interactive games, a web-server, ...
occam-\pi has been used successfully to create systems which could be described as ‘complex’

- operating-system components (the RMoX system)
- interactive games, a web-server, ...

- diagram showing relationships between console, kernel, fs.core, driver.core, and net.core.
occam-π has been used successfully to create systems which could be described as ‘complex’
- operating-system components (the RMoX system)
- interactive games, a web-server, ...

Fred Barnes, August 2006
occam-π has been used successfully to create systems which could be described as ‘complex’

- operating-system components (the RMoX system)
- interactive games, a web-server, ...

Fred Barnes, August 2006
occam-π has been used successfully to create systems which could be described as ‘complex’

- operating-system components (the RMoX system)
- interactive games, a web-server, ...

![Diagram of system components]

Fred Barnes, August 2006
occam-π has been used successfully to create systems which could be described as ‘complex’

- operating-system components (the RMoX system)
- interactive games, a web-server, ...

---

Fred Barnes, August 2006
occam-π has been used successfully to create systems which could be described as 'complex'

- operating-system components (the RMoX system)
- interactive games, a web-server, ...

Fred Barnes, August 2006
The Blood Clotting Model
We start with a simplified version of the models: 1-D pipeline of site processes

- a site is either empty or full, indicating that it contains a sticky platelet
- platelets are injected at one end and ‘flow’ down the pipeline
- when one bumps into another, they stick
The Blood Clotting Model

- We start with a simplified version of the models: 1-D pipeline of site processes
  - a site is either empty or full, indicating that it contains a sticky platelet
  - platelets are injected at one end and ‘flow’ down the pipeline
  - when one bumps into another, they stick

- Can experiment with the implementation by tweaking parameters such as the probability of a clot advancing and injection rate
The Blood Clotting Model

- We start with a simplified version of the models: 1-D pipeline of site processes
  - a site is either empty or full, indicating that it contains a sticky platelet
  - platelets are injected at one end and ‘flow’ down the pipeline
  - when one bumps into another, they stick

- Can experiment with the implementation by tweaking parameters such as the probability of a clot advancing and injection rate
We start with a simplified version of the models: 1-D pipeline of site processes:

- A site is either empty or full, indicating that it contains a sticky platelet.
- Platelets are injected at one end and ‘flow’ down the pipeline.
- When one bumps into another, they stick.

Can experiment with the implementation by tweaking parameters such as the probability of a clot advancing and injection rate.
We start with a simplified version of the models: 1-D pipeline of site processes

- a site is either empty or full, indicating that it contains a sticky platelet
- platelets are injected at one end and ‘flow’ down the pipeline
- when one bumps into another, they stick

Can experiment with the implementation by tweaking parameters such as the probability of a clot advancing and injection rate
We start with a simplified version of the models: 1-D pipeline of **site** processes

- a site is either empty or full, indicating that it contains a **sticky platelet**
- platelets are injected at one end and ‘flow’ down the pipeline
- when one bumps into another, they **stick**

Can experiment with the implementation by tweaking parameters such as the probability of a clot advancing and injection rate
We start with a simplified version of the models: 1-D pipeline of site processes
- a site is either empty or full, indicating that it contains a sticky platelet
- platelets are injected at one end and ‘flow’ down the pipeline
- when one bumps into another, they stick

Can experiment with the implementation by tweaking parameters such as the probability of a clot advancing and injection rate
We start with a simplified version of the models: 1-D pipeline of site processes

- a site is either empty or full, indicating that it contains a sticky platelet
- platelets are injected at one end and ‘flow’ down the pipeline
- when one bumps into another, they stick

Can experiment with the implementation by tweaking parameters such as the probability of a clot advancing and injection rate.
We start with a simplified version of the models: 1-D pipeline of site processes
- a site is either empty or full, indicating that it contains a sticky platelet
- platelets are injected at one end and ‘flow’ down the pipeline
- when one bumps into another, they stick

Can experiment with the implementation by tweaking parameters such as the probability of a clot advancing and injection rate
The Blood Clotting Model
The CSP model uses a series of SITE processes, synchronising on shared events to control progress.
The CSP model uses a series of SITE processes, synchronising on shared events to control progress.

\[
\text{EVENTS}(i) = \{\text{pass}.i, \text{pass}.i+1, \\
\quad \text{pass}.i+2, \text{tok}\}
\]

\[
\text{SYSTEM} = \parallel i:\{0..(N-1)\}
\]

\[@\text{EVENTS}(i)\] SITE(i)
The Blood Clotting Model

The CSP model uses a series of SITE processes, synchronising on shared events to control progress.

EVENTS(i) = \{pass.\(i\), pass.\(i+1\), pass.\(i+2\), tock\}

SYSTEM = || i:{0..(N-1)}
            @[EVENTS(i)] SITE(i)

SITE(i) = EMPTY(i)
The CSP model uses a series of SITE processes, synchronising on shared events to control progress.

\[
\text{EVENTS}(i) = \{\text{pass}.i, \text{pass}.i+1, \text{pass}.i+2, \text{trock}\}
\]

\[
\text{SYSTEM} = || i:0..(N-1) \]
\[
\text{@[EVENTS(i)] SITE}(i)
\]

\[
\text{SITE}(i) = \text{EMPTY}(i)
\]

\[
\text{EMPTY}(i) = \\
\text{pass}.i \rightarrow \text{ALMOST}(i) [ ] \\
\text{pass}.i+2 \rightarrow \text{EMPTY}(i) [ ] \\
\text{trock} \rightarrow \text{EMPTY}(i)
\]
The Blood Clotting Model

The CSP model uses a series of SITE processes, synchronising on shared events to control progress.

EVENTS(i) = \{pass.i, pass.i+1, pass.i+2, tock\}

SYSTEM = \| i:0..(N-1) \| @EVENTS(i) SITE(i)

SITE(i) = EMPTY(i)

EMPTY(i) =
- pass.i -> ALMOST(i) []
- pass.i+2 -> EMPTY(i) []
- tock -> EMPTY(i)

ALMOST(i) =
- pass.i+2 -> ALMOST(i) []
- tock -> FULL(i)
The Blood Clotting Model

The CSP model uses a series of SITE processes, synchronising on shared events to control progress.

EVENTS(i) = \{pass.i, pass.i+1, pass.i+2, tock\}
SYSTEM = \| i:{0..(N-1)}
\@[EVENTS(i)] SITE(i)

SITE(i) = EMPTY(i)

EMPTY(i) =
  pass.i \rightarrow\ ALMOST(i) []
  pass.i+2 \rightarrow\ EMPTY(i) []
  tock \rightarrow\ EMPTY(i)

ALMOST(i) =
  pass.i+2 \rightarrow\ ALMOST(i) []
  tock \rightarrow\ FULL(i)

FULL(i) =
  pass.i+1 \rightarrow\ EMPTY(i) []
  pass.i+2 \rightarrow\ pass.i+1 \rightarrow\ EMPTY(i) []
  tock \rightarrow\ FULL(i)
Implementation in occam-$\pi$
Almost a direct conversion, which could be done automatically
Implementation in occam-$\pi$

Almost a direct conversion, which could be done automatically

\[
\begin{align*}
\text{EVENTS}(i) &= \{\text{pass}.i, \text{pass}.i+1, \\
& \quad \text{pass}.i+2, \text{tock}\} \\
\text{SYSTEM} &= \parallel i:\{0..(N-1)\} \\
& \quad @[\text{EVENTS}(i)] \ \text{SITE}(i)
\end{align*}
\]
Almost a direct conversion, which could be done automatically

\[
EVENTS(i) = \{\text{pass}.i, \text{pass}.i+1, \\
\text{pass}.i+2, \text{tock}\}
\]

\[
\text{SYSTEM} = || i:\{0..(N-1)\} \\
@[EVENTS(i)] \text{SITE}(i)
\]

\[
[N+2]\text{ALT.BARRIER} \text{ pass:} \\
\text{ALT.BARRIER} \text{ tock:} \\
\text{PAR} i = 0 \text{ FOR N} \\
\text{site (pass}[i], \text{pass}[i+1], \\
\text{pass}[i+2], \text{tock})
\]
Implementation in occam-π

Almost a direct conversion, which could be done automatically

EVENTS(i) = \{pass.i, pass.i+1, pass.i+2, tock\}
SYSTEM = \| i:\{0..(N-1)\} @[EVENTS(i)] SITE(i)

SITE(i) = EMPTY(i)
EMPTY(i) =
    pass.i -> ALMOST(i) []
    pass.i+2 -> EMPTY(i) []
    tock -> EMPTY(i)

[N+2]ALT.BARRIER pass:
ALT.BARRIER tock:
PAR i = 0 FOR N
  site (pass[i], pass[i+1], pass[i+2], tock)
Almost a direct conversion, which could be done automatically

EVENTS(i) = \{pass.\,i, pass.\,i+1, pass.\,i+2, tock\}
SYSTEM = || i:\{0..(N-1)\}
          @\{EVENTS(i)\} SITE(i)

SITE(i) = EMPTY(i)
EMPTY(i) =
  pass.\,i \rightarrow ALMOST(i) []
  pass.\,i+2 \rightarrow EMPTY(i) []
  tock \rightarrow EMPTY(i)

\begin{align*}
\text{[N+2]ALT.BARRIER pass:} \\
\text{ALT.BARRIER tock:} \\
\text{PAR i = 0 FOR N} \\
\text{site (pass[i], pass[i+1], pass[i+2], tock)}
\end{align*}

\begin{align*}
\text{PROC site (ALT.BARRIER me, me.1, me.2, tock)} \\
\text{INITIAL INT state IS EMPTY:} \\
\text{WHILE TRUE} \\
\text{CASE state} \\
\quad \cdots \text{EMPTY case} \\
\quad \cdots \text{ALMOST case} \\
\quad \cdots \text{FULL case} \\
\end{align*}
Almost a direct conversion, which could be done automatically.

EVENTS(i) = \{pass.i, pass.i+1, pass.i+2, tock\}
SYSTEM = || i:{0..(N-1)}
@EVENTS(i) SITE(i)

SITE(i) = EMPTY(i)
EMPTY(i) =
  pass.i -> ALMOST(i) []
  pass.i+2 -> EMPTY(i) []
  tock -> EMPTY(i)

[N+2] ALT.BARRIER pass:
ALT.BARRIER tock:
PAR i = 0 FOR N
  site (pass[i], pass[i+1], pass[i+2], tock)

PROC site (ALT.BARRIER me, me.1, me.2, tock)
  INITIAL INT state IS EMPTY
  WHILE TRUE
    CASE state
      ... EMPTY case
      ... ALMOST case
      ... FULL case

    ALT
      SYNC me
      state := ALMOST
      SYNC me.2
      SKIP
      SYNC tock
      SKIP

(further details given in the paper)
Multiway Synchronisation
Implementations of this typically involve a **two-phase commit** protocol:

- Using a different **manager** process for each event.
- Managers receive offers at any time, counting down to zero.
- When all enrolled processes have offered, the manager starts to commit.
- Offers can be withdrawn at any time, including after zero has been reached.
Implementations of this typically involve a **two-phase commit** protocol

- using a different **manager** process for each event
- managers receive offers at any time, counting down to zero
- when all enrolled processes have offered, manager starts to commit
- offers can be withdrawn at any time, including after zero has been reached

Offers, withdrawals and commits fly around in parallel, two-phase protocol is needed to secure the operation — moderately expensive
Implementations of this typically involve a **two-phase commit** protocol

- using a different **manager** process for each event
- managers receive offers at any time, counting down to zero
- when all enrolled processes have offered, manager starts to commit
- offers can be withdrawn at any time, including after zero has been reached

Offers, withdrawals and commits fly around in parallel, two-phase protocol is needed to secure the operation — moderately expensive

Fred Barnes, August 2006
The Oracle Implementation
When explaining the system to a colleague at Kent, he wondered why all manager processes weren’t integrated into a single, serial, process dealing with offers one at a time — we wondered too!

- parallel offers, countdowns, cancel messages and collapses disappear
- something is lost, however: some viable choices may never be made because of the way in which the oracle process operates — still legal choices, however
When explaining the system to a colleague at Kent, he wondered why all manager processes weren’t integrated into a single, serial, process dealing with offers one at a time — we wondered too!

- parallel offers, countdowns, cancel messages and collapses disappear
- something is lost, however: some viable choices may never be made because of the way in which the oracle process operates — still legal choices, however
When explaining the system to a colleague at Kent, he wondered why all manager processes weren’t integrated into a single, serial, process dealing with offers one at a time — we wondered too!

- parallel offers, countdowns, cancel messages and collapses disappear
- something is lost, however: some viable choices may never be made because of the way in which the oracle process operates — still legal choices, however

Processes make all their offers at once, oracle responds with selected index

- internally maintains counters for each event, synchronising when those reach zero — no second stage needed
Inside the Oracle

- Events
- Counts
- Processes
- Offers
- Ask?
- Ans!
Inside the Oracle

events
counts

<table>
<thead>
<tr>
<th>processes</th>
<th>counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>m-1</td>
<td>42</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>m-1</td>
</tr>
</tbody>
</table>

offers

<table>
<thead>
<tr>
<th>offers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>n-1</td>
</tr>
</tbody>
</table>

1, [3, m-1]
2, [3, 4, m-1]

ask?

ans!
Inside the Oracle

<table>
<thead>
<tr>
<th>events</th>
<th>counts</th>
<th>processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0, 1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0, 1, 2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>1, 2, 3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2, 3, 4</td>
<td>4</td>
</tr>
<tr>
<td>m-1</td>
<td>0, 1, ..., n-1</td>
<td>n-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>offers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>n-1</td>
</tr>
</tbody>
</table>

1. [3, m-1]
2. [3, 4, m-1]
4. [4, 6, m-1]
Inside the Oracle

<table>
<thead>
<tr>
<th>Events</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>processes</td>
<td>offers</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0, 1</td>
<td>1</td>
</tr>
<tr>
<td>0, 1, 2</td>
<td>2</td>
</tr>
<tr>
<td>1, 2, 3</td>
<td>3</td>
</tr>
<tr>
<td>2, 3, 4</td>
<td>1</td>
</tr>
<tr>
<td>0, 1, ..., n-1</td>
<td>41</td>
</tr>
<tr>
<td>1, [3, m-1]</td>
<td></td>
</tr>
<tr>
<td>2, [3, 4, m-1]</td>
<td></td>
</tr>
<tr>
<td>4, [4, 6, m-1]</td>
<td></td>
</tr>
</tbody>
</table>
### Inside the Oracle

#### Processes and Counts

<table>
<thead>
<tr>
<th>Events</th>
<th>Counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0, 1</td>
</tr>
<tr>
<td>2</td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>3</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>4</td>
<td>2, 3, 4</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>m-1</td>
<td>0, 1, ..., n-1</td>
</tr>
</tbody>
</table>

#### Offers

<table>
<thead>
<tr>
<th>Ask?</th>
<th>Offers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[3, m-1]</td>
</tr>
<tr>
<td>2</td>
<td>[3, 4, m-1]</td>
</tr>
<tr>
<td>3</td>
<td>[3, 5, m-1]</td>
</tr>
<tr>
<td>4</td>
<td>[4, 6, m-1]</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Process Activities

- **Ask**: Processes send offers to other processes.
- **Ans**: Processes respond to offers by accepting or declining.

**Notes**:
- Processes 1, 2, and 3 are initiating the process.
- Each process has a specific range of offers to consider.
- The system handles these interactions to manage concurrency and synchronization.
Inside the Oracle

<table>
<thead>
<tr>
<th>processes</th>
<th>counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>m-1</td>
<td>43</td>
</tr>
</tbody>
</table>

Events:
- 0: 0
- 1: 0, 1
- 2: 0, 1, 2
- 3: 1, 2, 3
- 4: 2, 3, 4
- m-1: 0, 1, ..., n-1

Counts:
- 0: 1
- 1: 2
- 2: 3
- 3: 3
- 4: 2
- m-1: 43

Offers:
- 0: 4, 6, m-1
- 1: 3
- 2: 3
- 3: 3
- 4: 3

Ask?
- 1: [3, m-1]
- 2: [3, 4, m-1]
- 4: [4, 6, m-1]
- 3: [3, 5, m-1]

Ans!
- 3
- 3
- 3
Inside the Oracle

<table>
<thead>
<tr>
<th>processes</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>m-1</th>
<th>...</th>
<th>n-1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0, 1</td>
<td>0, 1, 2</td>
<td>1, 2, 3</td>
<td>2, 3, 4</td>
<td></td>
<td>0, 1, ..., n-1</td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>counts</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>processes</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>n-1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>1, [3, m-1]</td>
<td>2, [3, 4, m-1]</td>
<td>3, [3, 5, m-1]</td>
<td>4, [4, 6, m-1]</td>
<td>1, [3, m-1]</td>
</tr>
<tr>
<td>offers</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fred Barnes, August 2006
Inside the Oracle

<table>
<thead>
<tr>
<th>processes</th>
<th>offers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3, m-1</td>
</tr>
<tr>
<td>1</td>
<td>2, 4, m-1</td>
</tr>
<tr>
<td>2</td>
<td>5, m-1</td>
</tr>
<tr>
<td>3</td>
<td>4, 6, m-1</td>
</tr>
<tr>
<td>m-1</td>
<td>1, [3, m-1]</td>
</tr>
<tr>
<td></td>
<td>2, [3, 4, m-1]</td>
</tr>
<tr>
<td></td>
<td>4, [4, 6, m-1]</td>
</tr>
<tr>
<td></td>
<td>3, [3, 5, m-1]</td>
</tr>
<tr>
<td></td>
<td>1, [3, m-1]</td>
</tr>
<tr>
<td></td>
<td>2, [2, 4, m-1]</td>
</tr>
<tr>
<td></td>
<td>3, [5, m-1]</td>
</tr>
</tbody>
</table>

events

counts

<table>
<thead>
<tr>
<th>processes</th>
<th>events</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0, 1</td>
</tr>
<tr>
<td>1</td>
<td>0, 1</td>
</tr>
<tr>
<td>2</td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>3</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>4</td>
<td>2, 3, 4</td>
</tr>
<tr>
<td>m-1</td>
<td>0, 1, ..., n-1</td>
</tr>
</tbody>
</table>

ask?

ans!
Inside the Oracle

<table>
<thead>
<tr>
<th>processes</th>
<th>counts</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>m-1</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>offers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>n-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>asks?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, [3, m-1]</td>
</tr>
<tr>
<td>2, [3, 4, m-1]</td>
</tr>
<tr>
<td>4, [4, 6, m-1]</td>
</tr>
<tr>
<td>3, [3, 5, m-1]</td>
</tr>
<tr>
<td>1, [3, m-1]</td>
</tr>
<tr>
<td>2, [2, 4, m-1]</td>
</tr>
<tr>
<td>3, [5, m-1]</td>
</tr>
</tbody>
</table>

... and so on
Beyond the Oracle (1)
Beyond the Oracle (1)

There main drawback of the oracle technique is the explicit plumbing required

- the ideal conversion uses the ALT language binding
Beyond the Oracle (1)

- There main drawback of the oracle technique is the explicit plumbing required
  - the ideal conversion uses the ALT language binding

- A modified BARRIER type is under construction in a new occam-π compiler
  - based on a decentralised version of the oracle logic
  - would provide a sufficient implementation for the ALT.BARRIER in the paper
Beyond the Oracle (1)

- There main drawback of the oracle technique is the explicit plumbing required
  - the ideal conversion uses the ALT language binding

- A modified BARRIER type is under construction in a new occam-π compiler
  - based on a decentralised version of the oracle logic
  - would provide a sufficient implementation for the ALT.BARRIER in the paper

<table>
<thead>
<tr>
<th>event structure</th>
</tr>
</thead>
</table>
| int ref_count            | 3  
| int enroll_count         | 2  
| int down_count           | 2  
| waitq_t *fptr            |  
| waitq_t *bptr            |  

There main drawback of the oracle technique is the explicit plumbing required
- the ideal conversion uses the ALT language binding

A modified BARRIER type is under construction in a new occam-π compiler
- based on a decentralised version of the oracle logic
- would provide a sufficient implementation for the ALT.BARRIER in the paper

```
322
(wait-queue)

(int ref_count, 3)
(int enroll_count, 2)
(int down_count, 1)
(waitq_t *fptr)
(waitq_t *bptr)

(waitq_t *next)
(uint *wptr)
(uint pri)

(waitq) structures allocated dynamically as processes offer
```

(alting process)
There main drawback of the oracle technique is the explicit plumbing required
- the ideal conversion uses the ALT language binding

A modified BARRIER type is under construction in a new occam-π compiler
- based on a decentralised version of the oracle logic
- would provide a sufficient implementation for the ALT.BARRIER in the paper

Because a process may be waiting on multiple events, must force all disabling
sequences to happen when synchronisation is complete (before any more
multiway syncs can begin) — done with a simple global semaphore
- and some extra logic in ALT enabling/disabling
Beyond the Oracle (2)
Beyond the Oracle (2)

- Functional implementation of multiway synchronisations, but some drawbacks
  - dynamic allocation and free of the wait-queue
  - does not support *interleaving* or *partial synchronisation*
Beyond the Oracle (2)

- Functional implementation of multiway synchronisations, but some drawbacks
  - dynamic allocation and free of the wait-queue
  - does not support **interleaving** or **partial synchronisation**

- However, now implementing another one which does  :-)
  - requires some non-trivial compiler support to allocate data structures
Beyond the Oracle (2)

- Functional implementation of multiway synchronisations, but some drawbacks
  - dynamic allocation and free of the wait-queue
  - does not support **interleaving** or **partial synchronisation**

- However, now implementing another one which does :-)
  - requires some non-trivial compiler support to allocate data structures

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>barrier</td>
<td></td>
</tr>
<tr>
<td>int sets_enrolled</td>
<td>0</td>
</tr>
<tr>
<td>int sets_downcount</td>
<td>0</td>
</tr>
<tr>
<td>parbar_t *set_fptr</td>
<td></td>
</tr>
<tr>
<td>parbar_t *set_bptr</td>
<td></td>
</tr>
</tbody>
</table>
Beyond the Oracle (2)

- Functional implementation of multiway synchronisations, but some drawbacks
  - dynamic allocation and free of the wait-queue
  - does not support *interleaving* or *partial synchronisation*

- However, now implementing another one which does :-)
  - requires some non-trivial compiler support to allocate data structures
Beyond the Oracle (2)

- Functional implementation of multiway synchronisations, but some drawbacks
  - dynamic allocation and free of the wait-queue
  - does not support **interleaving** or **partial synchronisation**

However, now implementing another one which does :-)
- requires some non-trivial compiler support to allocate data structures
Beyond the Oracle (2)

- Functional implementation of multiway synchronisations, but some drawbacks
  - dynamic allocation and free of the wait-queue
  - does not support **interleaving** or **partial synchronisation**

- However, now implementing another one which does :-
  - requires some non-trivial compiler support to allocate data structures
Beyond the Oracle (2)

- Functional implementation of multiway synchronisations, but some drawbacks
  - dynamic allocation and free of the wait-queue
  - does not support **interleaving** or **partial synchronisation**

- However, now implementing another one which does :-)
  - requires some non-trivial compiler support to allocate data structures

<table>
<thead>
<tr>
<th>barrier</th>
<th>par-barrier</th>
<th>proc-barrier</th>
</tr>
</thead>
<tbody>
<tr>
<td>int sets_enrolled</td>
<td>1</td>
<td>procbar_t *q_next</td>
</tr>
<tr>
<td>int sets_downcount</td>
<td>1</td>
<td>procbar_t *q_prev</td>
</tr>
<tr>
<td>parbar_t *set_fptr</td>
<td></td>
<td>parbar_t *pbar_link</td>
</tr>
<tr>
<td>parbar_t *set_bptr</td>
<td></td>
<td>uint *wpitr</td>
</tr>
</tbody>
</table>

sync-count specifies how many processes required to offer in this set
Beyond the Oracle (2)

• Functional implementation of multiway synchronizations, but some drawbacks
  • dynamic allocation and free of the wait-queue
  • does not support **interleaving** or **partial synchronisation**

• However, now implementing another one which does :-(
  • requires some non-trivial compiler support to allocate data structures

---

**barrier**

- int sets_enrolled
- int sets_downcount
- parbar_t *set_fptr
- parbar_t *set_bptr

**par-barrier**

- parbar_t *next_set
- parbar_t *prev_set
- parbar_t *parent_set
- bar_t *bar_link
- int enroll_count
- int sync_count
- int down_count
- procbar_t *q_fptr
- procbar_t *q_bptr

**proc-barrier**

- procbar_t *q_next
- procbar_t *q_prev
- parbar_t *pbar_link
- uint *wptr
- int alt_flags

**sync-count** specifies how many processes required to offer in this set

**down-count** may drop below zero, indicating surplus; does sync as it goes past
Beyond the Oracle (2)
Implementation of this in the run-time system is mostly complete

- still developing the compiler support for handling it:
  mostly allocating data structures in the right places for processes
- will also be used by another mechanism in the same compiler:
  direct compilation of CSP (to be presented at CPA-2006)
Beyond the Oracle (2)

- Implementation of this in the run-time system is mostly complete
  - still developing the compiler support for handling it:
    mostly allocating data structures in the right places for processes
  - will also be used by another mechanism in the same compiler:
    direct compilation of CSP (to be presented at CPA-2006)

- Provides a nice solution to the classic *santa-claus* problem, still thinking about
  the language binding:
Implementation of this in the run-time system is mostly complete

- still developing the compiler support for handling it:
  mostly allocating data structures in the right places for processes
- will also be used by another mechanism in the same compiler:
  direct compilation of CSP (to be presented at CPA-2006)

Provides a nice solution to the classic **santa-claus** problem, still thinking about the language binding:

```plaintext
BARRIER e.sync:
BARRIER r.sync:
PAR
  santa (e.sync, r.sync)
  PAR i = 0 FOR 9
    reindeer (r.sync)
  PAR i = 0 FOR 10 INTERLEAVE e.sync(3)
  elf (e.sync)
```
Implementation of this in the run-time system is mostly complete

- still developing the compiler support for handling it:
  mostly allocating data structures in the right places for processes
- will also be used by another mechanism in the same compiler:
  direct compilation of CSP (to be presented at CPA-2006)

Provides a nice solution to the classic **santa-claus** problem, still thinking about the language binding:

```plaintext
PAR
    santa (e.sync, r.sync)
PAR i = 0 FOR 9
    reindeer (r.sync)
PAR i = 0 FOR 10 INTERLEAVE e.sync(3)
    elf (e.sync)
PROC santa (BARRIER elves, reindeer)
    WHILE TRUE
    PRI ALT
        elves
            ... meet with elves
        reindeer
            ... go deliver presents
```
Conclusions
The work presents aspects of the occam-$\pi$ simulation of the TUNA platelet case study, with a novel and fast mechanism for resolving CSP external choice between multiway synchronisation events from which any participant may withdraw its offer at any time.

- efficient implementation of this in occam-$\pi$, the developments of which enable the simulation of more complex clotting models involving interleaving.
The work presents aspects of the occam-π simulation of the TUNA platelet case study, with a novel and fast mechanism for resolving CSP external choice between multiway synchronisation events from which any participant may withdraw its offer at any time.

- efficient implementation of this in occam-π, the developments of which enable the simulation of more complex clotting models involving interleaving

Using the behaviour of this model as a comparison for other occam-π implementations — already have efficient lazy simulations.
Conclusions

The work presents aspects of the occam-\(\pi\) simulation of the TUNA platelet case study, with a novel and fast mechanism for resolving CSP external choice between multiway synchronisation events from which any participant may withdraw its offer at any time.

- efficient implementation of this in occam-\(\pi\), the developments of which enable the simulation of more complex clotting models involving interleaving

Using the behaviour of this model as a comparison for other occam-\(\pi\) implementations — already have efficient lazy simulations

Towards a better understanding of the rules and requirements for nanotech simulations

Beyond that, other biological, chemical and physical simulations
Conclusions

The work presents aspects of the occam-π simulation of the TUNA platelet case study, with a novel and fast mechanism for resolving CSP **external choice** between **multiway synchronisation events** from which any participant may withdraw its offer at any time

- efficient implementation of this in occam-π, the developments of which enable the simulation of more complex clotting models involving **interleaving**

Using the behaviour of this model as a comparison for other occam-π implementations — already have efficient **lazy simulations**

Towards a better understanding of the rules and requirements for **nanotech** simulations

Beyond that, other biological, chemical and physical simulations
Conclusions

- The work presents aspects of the occam-π simulation of the TUNA platelet case study, with a novel and fast mechanism for resolving CSP **external choice** between **multiway synchronisation events** from which any participant may withdraw its offer at any time
  - efficient implementation of this in occam-π, the developments of which enable the simulation of more complex clotting models involving **interleaving**

- Using the behaviour of this model as a comparison for other occam-π implementations — already have efficient **lazy simulations**

- Towards a better understanding of the rules and requirements for **nanotech** simulations

- Beyond that, other biological, chemical and physical simulations
Conclusions

- The work presents aspects of the occam-$\pi$ simulation of the TUNA platelet case study, with a novel and fast mechanism for resolving CSP external choice between multiway synchronisation events from which any participant may withdraw its offer at any time.
  - efficient implementation of this in occam-$\pi$, the developments of which enable the simulation of more complex clotting models involving interleaving
- Using the behaviour of this model as a comparison for other occam-$\pi$ implementations — already have efficient lazy simulations
- Towards a better understanding of the rules and requirements for nanotech simulations
- Beyond that, other biological, chemical and physical simulations
Conclusions

The work presents aspects of the occam-π simulation of the TUNA platelet case study, with a novel and fast mechanism for resolving CSP external choice between multiway synchronisation events from which any participant may withdraw its offer at any time.

- efficient implementation of this in occam-π, the developments of which enable the simulation of more complex clotting models involving interleaving.

Using the behaviour of this model as a comparison for other occam-π implementations — already have efficient lazy simulations.

Towards a better understanding of the rules and requirements for nanotech simulations.

Beyond that, other biological, chemical and physical simulations.

Fred Barnes, August 2006
Conclusions

The work presents aspects of the occam-π simulation of the TUNA platelet case study, with a novel and fast mechanism for resolving CSP **external choice** between **multiway synchronisation events** from which any participant may withdraw its offer at any time

- efficient implementation of this in occam-π, the developments of which enable the simulation of more complex clotting models involving **interleaving**

Using the behaviour of this model as a comparison for other occam-π implementations — already have efficient **lazy simulations**

Towards a better understanding of the rules and requirements for **nanotech** simulations

Beyond that, other biological, chemical and physical simulations
This work was carried out as part of the EPSRC funded TUNA project (EP/C516966/1), between the Universities of York, Surrey and Kent:

- http://www.cs.york.ac.uk/nature/tuna/index.htm

Information about occam-$\pi$ can be found at:

- http://www.occam-pi.org/

Information about the KRoC occam-$\pi$ system and new compiler can be found at:

- http://www.cs.kent.ac.uk/projects/ofa/kroc/
- http://www.cs.kent.ac.uk/projects/ofa/nocc/

The TUNA project team:

**York**: Ana Cavalcanti, Heather Turner, Susan Stepney, Jim Woodcock

**Surrey**: Helen Treharne, Steve Schneider

**Kent**: Fred Barnes, Peter Welch