

Checking Process-Oriented Operating System Behaviour using CSP and Refinement

PLOS Workshop, Big Sky, 11th October 2009

Fred Barnes and Carl Ritson

Programming Languages and Systems Research Group
School of Computing, University of Kent, Canterbury

`{F.R.M.Barnes,C.G.Ritson}@kent.ac.uk`

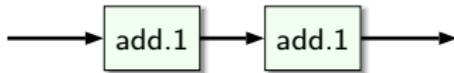
`http://www.cs.kent.ac.uk/~{frmb,cgr}/`

Process-Orientation and $\text{occam-}\pi$

- **Process-orientation**: systems built from **concurrent processes** that **communicate** through **channels**.
- Channels uni-directional and strongly typed – **synchronous** communication.
- Processes organised into **layered networks** – structure, reuseability.
- **occam- π** is a language embodying these concepts.
 - derived from traditional occam (INMOS transputer, **CSP** algebra).
 - incorporates ideas of **mobility** from Milner's π -calculus – not limited to static networks of processes.
- Dynamic **reconfiguration** of process networks.
 - **moving** channel-ends around.
- Strict parallel-usage and alias checking – no unexpected **race-hazard** errors.

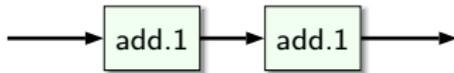
Process-Orientation and occam- π

- **Process-orientation**: systems built from **concurrent processes** that **communicate** through **channels**.
- Channels uni-directional and strongly typed – **synchronous** communication.
- Processes organised into **layered networks** – structure, reuseability.
- **occam- π** is a language embodying these concepts.
 - derived from traditional occam (INMOS transputer, **CSP** algebra).
 - incorporates ideas of **mobility** from Milner's π -calculus – not limited to static networks of processes.
- Dynamic **reconfiguration** of process networks.
 - **moving** channel-ends around.
- Strict parallel-usage and alias checking – no unexpected **race-hazard** errors.



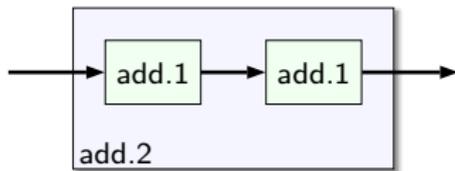
Process-Orientation and occam- π

- **Process-orientation**: systems built from **concurrent processes** that **communicate** through **channels**.
- Channels uni-directional and strongly typed – **synchronous** communication.
- Processes organised into **layered networks** – structure, reuseability.
- **occam- π** is a language embodying these concepts.
 - derived from traditional occam (INMOS transputer, **CSP** algebra).
 - incorporates ideas of **mobility** from Milner's π -calculus – not limited to static networks of processes.
- Dynamic **reconfiguration** of process networks.
 - **moving** channel-ends around.
- Strict parallel-usage and alias checking – no unexpected **race-hazard** errors.



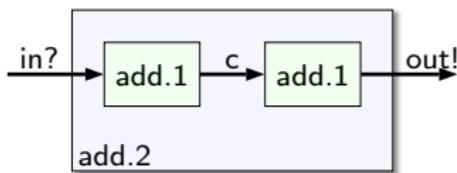
Process-Oriented and $\text{occam-}\pi$

- **Process-orientation**: systems built from **concurrent processes** that **communicate** through **channels**.
- Channels uni-directional and strongly typed – **synchronous** communication.
- Processes organised into **layered networks** – structure, reuseability.
- **occam- π** is a language embodying these concepts.
 - derived from traditional occam (INMOS transputer, **CSP** algebra).
 - incorporates ideas of **mobility** from Milner's π -calculus – not limited to static networks of processes.
- Dynamic **reconfiguration** of process networks.
 - **moving** channel-ends around.
- Strict parallel-usage and alias checking – no unexpected **race-hazard** errors.



Process-Oriented and occam- π

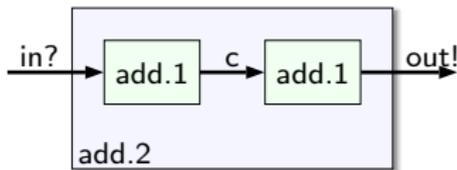
- **Process-orientation**: systems built from **concurrent processes** that **communicate** through **channels**.
- Channels uni-directional and strongly typed – **synchronous** communication.
- Processes organised into **layered networks** – structure, reuseability.



- **occam- π** is a language embodying these concepts.
 - derived from traditional occam (INMOS transputer, **CSP** algebra).
 - incorporates ideas of **mobility** from Milner's π -calculus – not limited to static networks of processes.
- Dynamic **reconfiguration** of process networks.
 - **moving** channel-ends around.
- Strict parallel-usage and alias checking – no unexpected **race-hazard** errors.

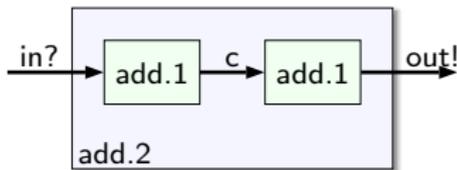
Process-Oriented and $\text{occam-}\pi$

- **Process-orientation**: systems built from **concurrent processes** that **communicate** through **channels**.
- Channels uni-directional and strongly typed – **synchronous** communication.
- Processes organised into **layered networks** – structure, reuseability.
- **occam- π** is a language embodying these concepts.
 - derived from traditional occam (INMOS transputer, **CSP** algebra).
 - incorporates ideas of **mobility** from Milner's π -calculus – not limited to static networks of processes.
- Dynamic **reconfiguration** of process networks.
 - **moving** channel-ends around.
- Strict parallel-usage and alias checking – no unexpected **race-hazard** errors.



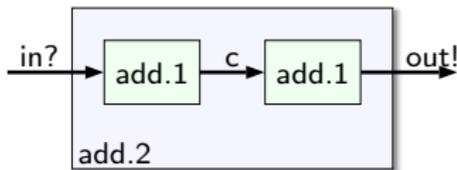
Process-Oriented and $\text{occam-}\pi$

- **Process-orientation**: systems built from **concurrent processes** that **communicate** through **channels**.
- Channels uni-directional and strongly typed – **synchronous** communication.
- Processes organised into **layered networks** – structure, reusability.
- **occam- π** is a language embodying these concepts.
 - derived from traditional occam (INMOS transputer, **CSP** algebra).
 - incorporates ideas of **mobility** from Milner's π -calculus – not limited to static networks of processes.
- Dynamic **reconfiguration** of process networks.
 - **moving** channel-ends around.
- Strict parallel-usage and alias checking – no unexpected **race-hazard** errors.



Process-Oriented and $\text{occam-}\pi$

- **Process-orientation**: systems built from **concurrent processes** that **communicate** through **channels**.
- Channels uni-directional and strongly typed – **synchronous** communication.
- Processes organised into **layered networks** – structure, reuseability.
- **occam- π** is a language embodying these concepts.
 - derived from traditional occam (INMOS transputer, **CSP** algebra).
 - incorporates ideas of **mobility** from Milner's π -calculus – not limited to static networks of processes.
- Dynamic **reconfiguration** of process networks.
 - **moving** channel-ends around.
- Strict parallel-usage and alias checking – no unexpected **race-hazard** errors.



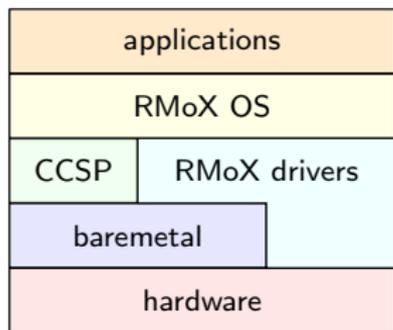
The RMoX Operating System

- An OS built on these ideas of **processes**, **channels** and **dynamic** network reconfiguration.
 - contains hundreds of concurrent processes, potentially thousands.
- Compiler and run-time system from the **KRoC**.
 - efficient scheduling and communication on multicore hardware.

■ Initial structure ...

The RMoX Operating System

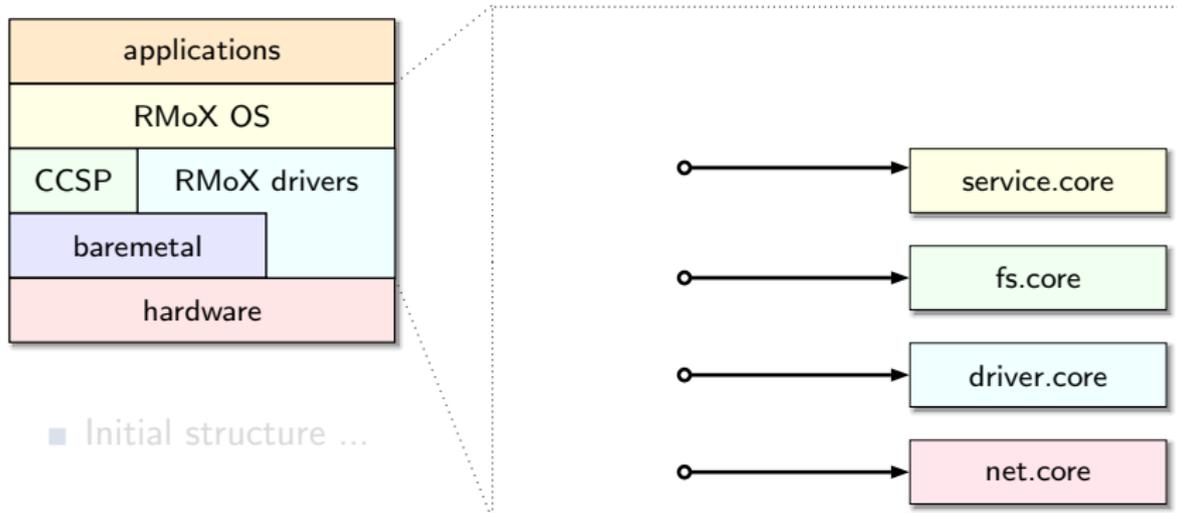
- An OS built on these ideas of **processes**, **channels** and **dynamic** network reconfiguration.
 - contains hundreds of concurrent processes, potentially thousands.
- Compiler and run-time system from the **KRoC**.
 - efficient scheduling and communication on multicore hardware.



- Initial structure ...

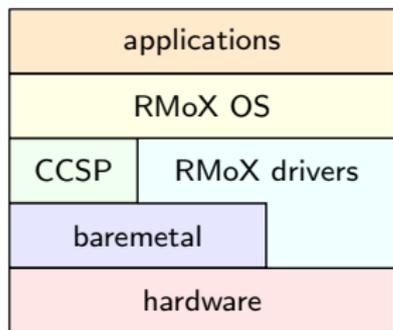
The RMoX Operating System

- An OS built on these ideas of **processes**, **channels** and **dynamic** network reconfiguration.
 - contains hundreds of concurrent processes, potentially thousands.
- Compiler and run-time system from the **KRoC**.
 - efficient scheduling and communication on multicore hardware.

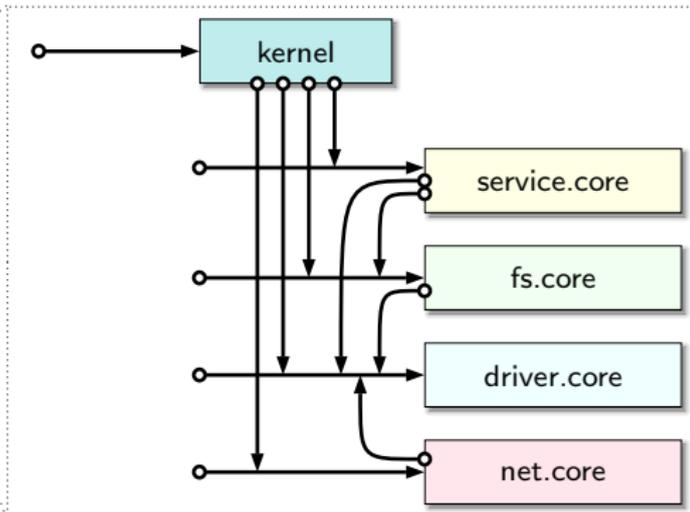


The RMoX Operating System

- An OS built on these ideas of **processes**, **channels** and **dynamic** network reconfiguration.
 - contains hundreds of concurrent processes, potentially thousands.
- Compiler and run-time system from the **KRoC**.
 - efficient scheduling and communication on multicore hardware.

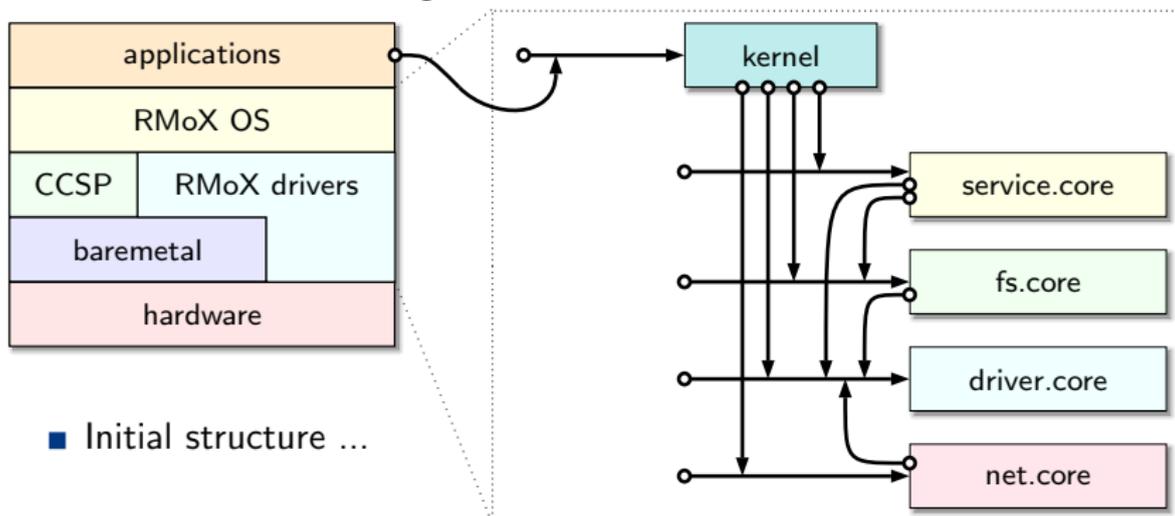


- Initial structure ...



The RMoX Operating System

- An OS built on these ideas of **processes**, **channels** and **dynamic** network reconfiguration.
 - contains hundreds of concurrent processes, potentially thousands.
- Compiler and run-time system from the **KRoC**.
 - efficient scheduling and communication on multicore hardware.



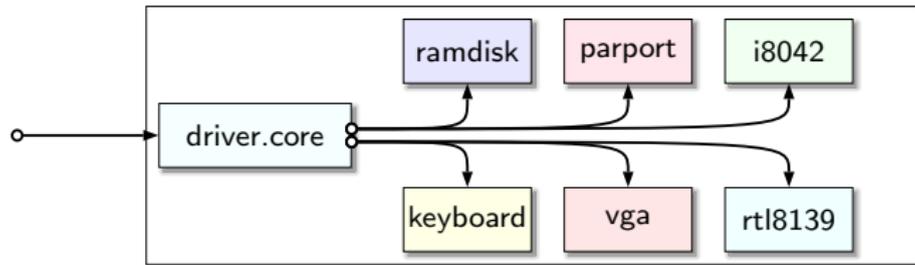
Dynamically Evolving Process Networks

- The RMoX process network **evolves** over time.
 - new processes dynamically created (spawned), old ones shut down.
 - connections between processes established to reflect operation.

- Organisation at this level is **client-server**.
 - guarantees of deadlock freedom (at this level).
 - cycles broken through the use of dynamically spawned processes.

Dynamically Evolving Process Networks

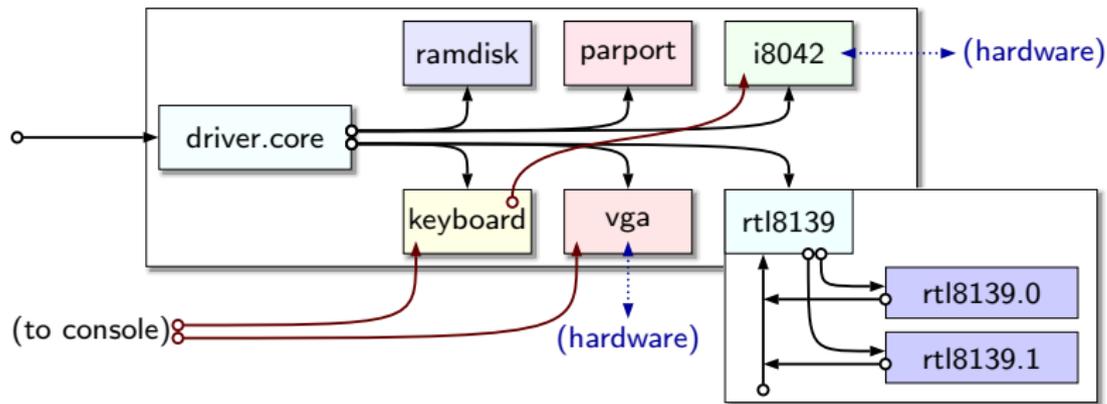
- The RMoX process network **evolves** over time.
 - new processes dynamically created (spawned), old ones shut down.
 - connections between processes established to reflect operation.



- Organisation at this level is **client-server**.
 - guarantees of deadlock freedom (at this level).
 - cycles broken through the use of dynamically spawned processes.

Dynamically Evolving Process Networks

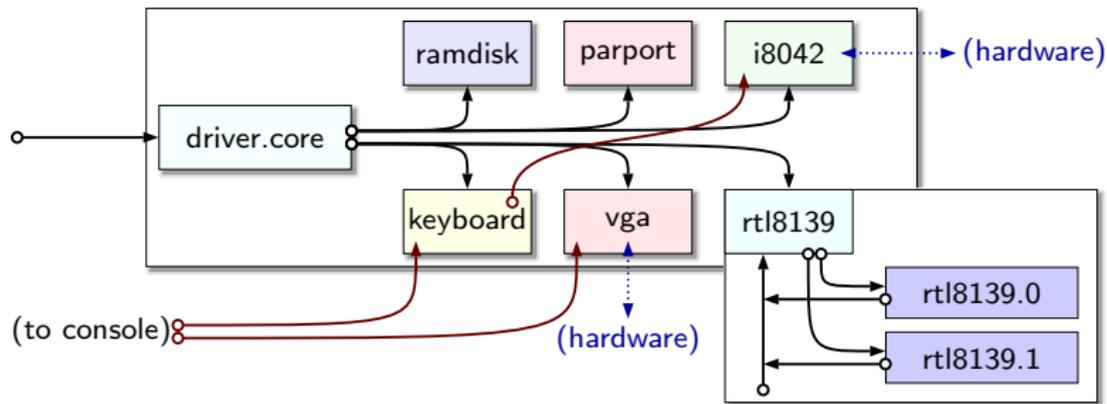
- The RMoX process network **evolves** over time.
 - new processes dynamically created (spawned), old ones shut down.
 - connections between processes established to reflect operation.



- Organisation at this level is **client-server**.
 - guarantees of deadlock freedom (at this level).
 - cycles broken through the use of dynamically spawned processes.

Dynamically Evolving Process Networks

- The RMoX process network **evolves** over time.
 - new processes dynamically created (spawned), old ones shut down.
 - connections between processes established to reflect operation.



- Organisation at this level is **client-server**.
 - guarantees of deadlock freedom (at this level).
 - cycles broken through the use of dynamically spawned processes.

A Flavour of occam- π

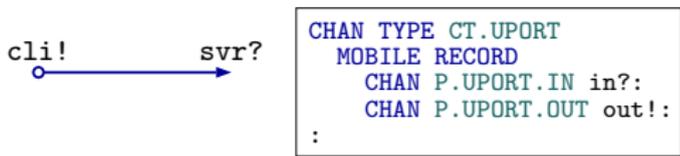
- Connections shown are **structured channel-types** (bundles).
 - a variety of **strongly typed** protocols are used to carry messages.

```
CHAN TYPE CT.UPORT
MOBILE RECORD
  CHAN P.UPORT.IN in?:
  CHAN P.UPORT.OUT out!:
```

- Code such as this can fail at run-time.
 - server sends an 'error' response, not handled by the client.
 - results in **deadlock** or a run-time error.

A Flavour of occam- π

- Connections shown are **structured channel-types** (bundles).
 - a variety of **strongly typed** protocols are used to carry messages.

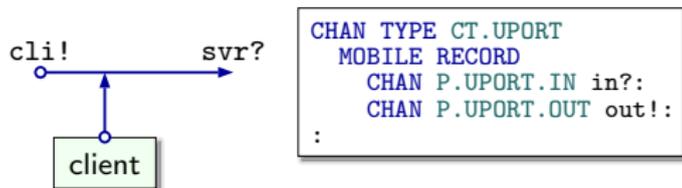


```
SHARED CT.UPORT! cli:
CT.UPORT? svr:
SEQ
cli, svr := MOBILE CT.UPORT
```

- Code such as this can fail at run-time.
 - server sends an 'error' response, not handled by the client.
 - results in **deadlock** or a run-time error.

A Flavour of occam- π

- Connections shown are **structured channel-types** (bundles).
 - a variety of **strongly typed** protocols are used to carry messages.



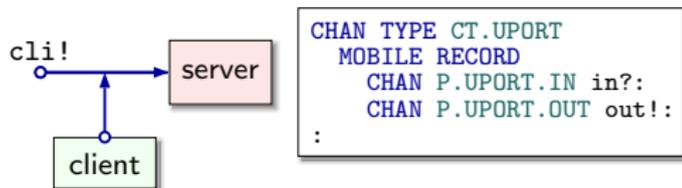
```

SHARED CT.UPORT! cli:
CT.UPORT? svr:
SEQ
cli, svr := MOBILE CT.UPORT
FORK client (cli!)
  
```

- Code such as this can fail at run-time.
 - server sends an 'error' response, not handled by the client.
 - results in **deadlock** or a run-time error.

A Flavour of occam- π

- Connections shown are **structured channel-types** (bundles).
 - a variety of **strongly typed** protocols are used to carry messages.



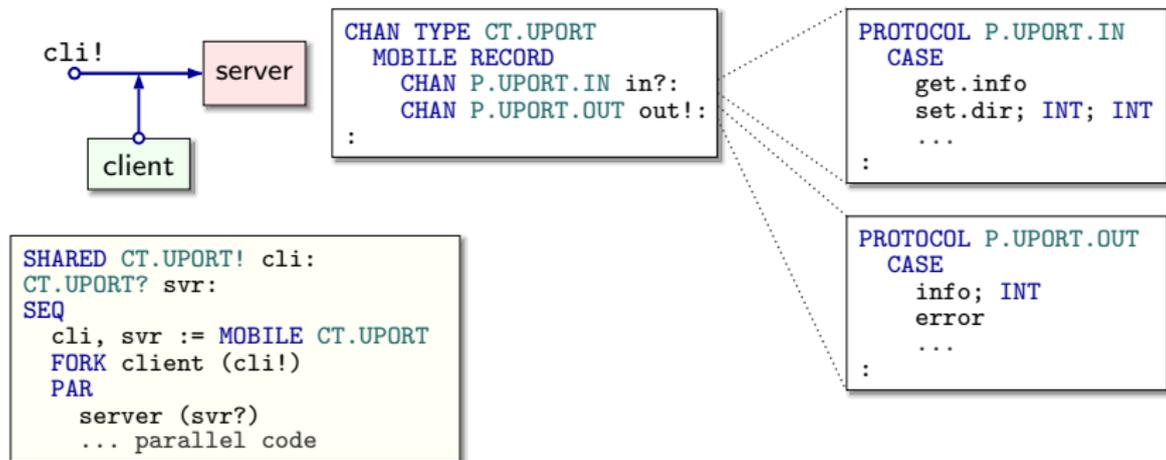
```

SHARED CT.UPORT! cli:
CT.UPORT? svr:
SEQ
  cli, svr := MOBILE CT.UPORT
  FORK client (cli!)
  PAR
    server (svr?)
    ... parallel code
  
```

- Code such as this can fail at run-time.
 - server sends an 'error' response, not handled by the client.
 - results in **deadlock** or a run-time error.

A Flavour of occam- π

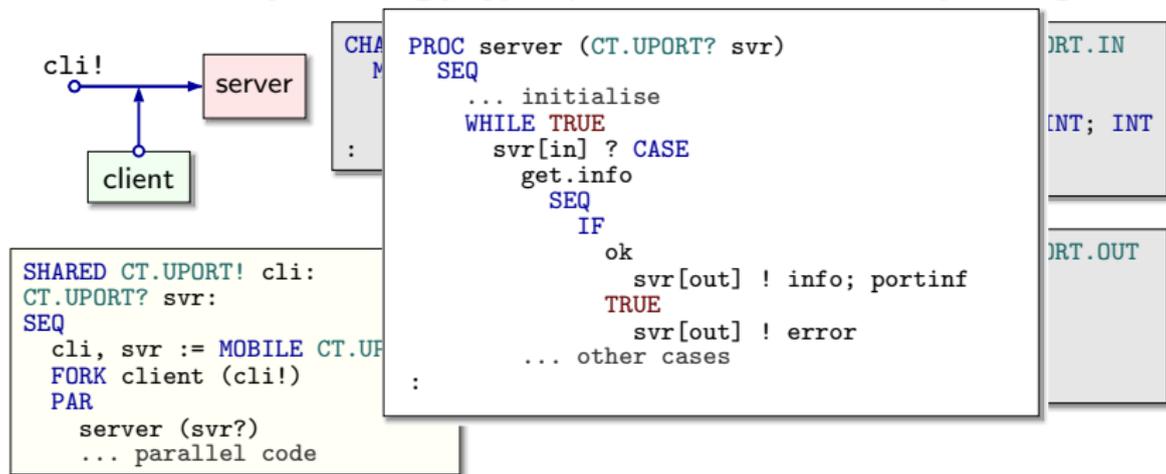
- Connections shown are **structured channel-types** (bundles).
 - a variety of **strongly typed** protocols are used to carry messages.



- Code such as this can fail at run-time.
 - server sends an 'error' response, not handled by the client.
 - results in **deadlock** or a run-time error.

A Flavour of occam- π

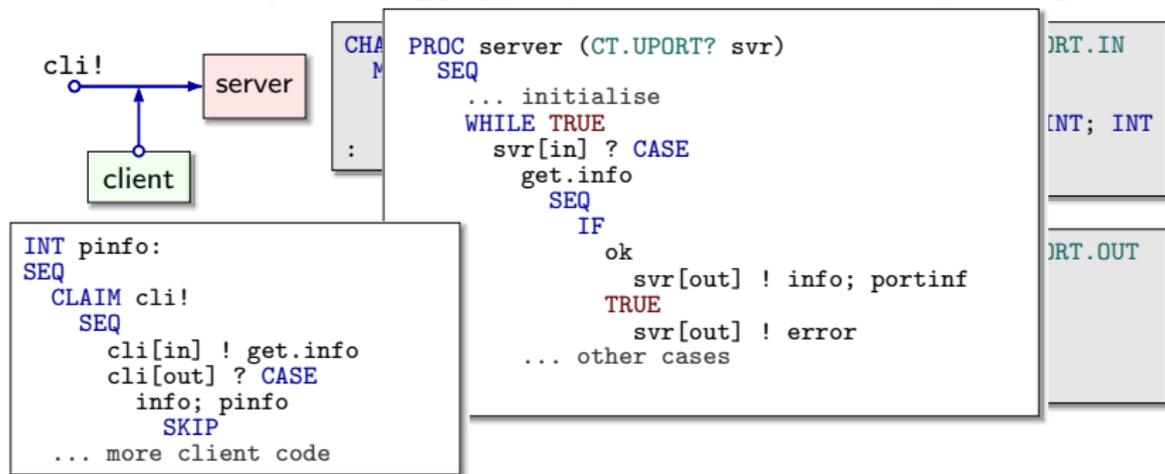
- Connections shown are **structured channel-types** (bundles).
 - a variety of **strongly typed** protocols are used to carry messages.



- Code such as this can fail at run-time.
 - server sends an 'error' response, not handled by the client.
 - results in **deadlock** or a run-time error.

A Flavour of occam- π

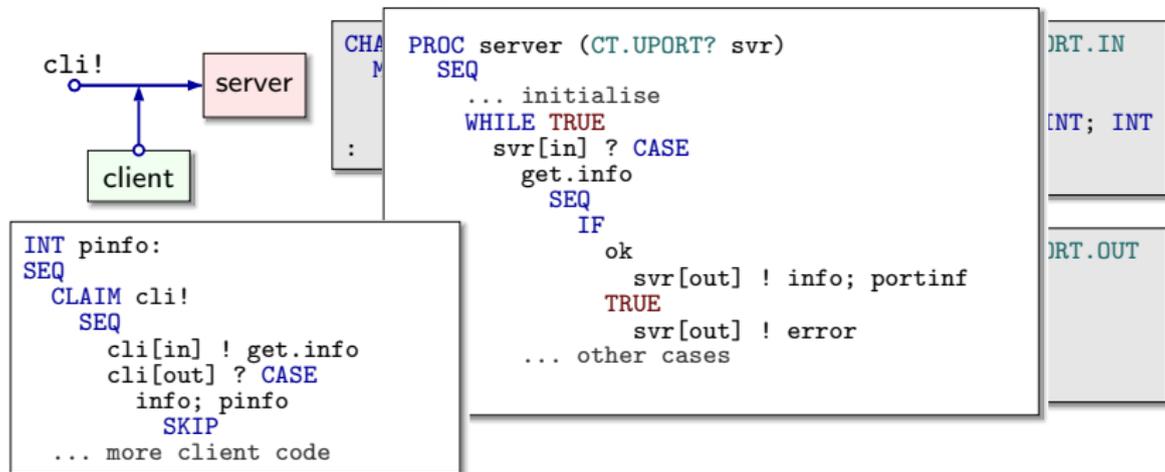
- Connections shown are **structured channel-types** (bundles).
 - a variety of **strongly typed** protocols are used to carry messages.



- Code such as this can fail at run-time.
 - server sends an 'error' response, not handled by the client.
 - results in **deadlock** or a run-time error.

A Flavour of occam- π

- Connections shown are **structured channel-types** (bundles).
 - a variety of **strongly typed** protocols are used to carry messages.



- Code such as this can fail at run-time.
 - server sends an 'error' response, not handled by the client.
 - results in **deadlock** or a run-time error.

Challenges

- Such errors relate to **improper use** of channel communication.
 - mostly resulting in deadlock (contagious).
 - most of the bugs discovered in RMoX (to date) relate to this.
- Extensive testing is **insufficient** for many (particularly embedded) systems, though it is a start.
 - not limited to RMoX: complex systems simulations developed as part of CoSMoS, and more generally, any **process-oriented** system.
 - for RMoX in particular, correct operation of **third-party** components.
- Proposed solution:
 - to check that processes **behave correctly** with respect to their **interfaces** (channel-bundle ends).
 - and to check the safe **composition** of processes and interfaces.
 - achieved through the **automatic generation** and **checking** of formal models of occam- π processes (using CSP and refinement).
- Not a **total solution**, currently:
 - level of abstraction, and correspondingly, complexity of the analysis.

Challenges

- Such errors relate to **improper use** of channel communication.
 - mostly resulting in deadlock (contagious).
 - most of the bugs discovered in RMoX (to date) relate to this.
- Extensive testing is **insufficient** for many (particularly embedded) systems, though it is a start.
 - not limited to RMoX: complex systems simulations developed as part of CoSMoS, and more generally, any **process-oriented** system.
 - for RMoX in particular, correct operation of **third-party** components.
- Proposed solution:
 - to check that processes **behave correctly** with respect to their **interfaces** (channel-bundle ends).
 - and to check the safe **composition** of processes and interfaces.
 - achieved through the **automatic generation** and **checking** of formal models of occam- π processes (using CSP and refinement).
- Not a **total solution**, currently:
 - level of abstraction, and correspondingly, complexity of the analysis.

Challenges

- Such errors relate to **improper use** of channel communication.
 - mostly resulting in deadlock (contagious).
 - most of the bugs discovered in RMoX (to date) relate to this.
- Extensive testing is **insufficient** for many (particularly embedded) systems, though it is a start.
 - not limited to RMoX: complex systems simulations developed as part of CoSMoS, and more generally, any **process-oriented** system.
 - for RMoX in particular, correct operation of **third-party** components.
- Proposed solution:
 - to check that processes **behave correctly** with respect to their **interfaces** (channel-bundle ends).
 - and to check the safe **composition** of processes and interfaces.
 - achieved through the **automatic generation** and **checking** of formal models of occam- π processes (using CSP and refinement).
- Not a **total solution**, currently:
 - level of abstraction, and correspondingly, complexity of the analysis.

Challenges

- Such errors relate to **improper use** of channel communication.
 - mostly resulting in deadlock (contagious).
 - most of the bugs discovered in RMoX (to date) relate to this.
- Extensive testing is **insufficient** for many (particularly embedded) systems, though it is a start.
 - not limited to RMoX: complex systems simulations developed as part of CoSMoS, and more generally, any **process-oriented** system.
 - for RMoX in particular, correct operation of **third-party** components.
- Proposed solution:
 - to check that processes **behave correctly** with respect to their **interfaces** (channel-bundle ends).
 - and to check the safe **composition** of processes and interfaces.
 - achieved through the **automatic generation** and **checking** of formal models of occam- π processes (using CSP and refinement).
- Not a **total solution**, currently:
 - level of abstraction, and correspondingly, complexity of the analysis.

The CSP Process Algebra

- Developed by **Hoare**, 1978, 1985.
 - used to model and reason about interactions between parallel processes (arbitrary numbers).
 - fundamental interaction is **synchronisation** on **events**.
- Has both **operational** and **denotational** semantics, the latter providing for formal reasoning, including of process composition.
 - semantic models are **traces**, **failures** and **divergences**.
- For example:

$$ADL = in \rightarrow c \rightarrow ADL$$

$$ADR = c \rightarrow out \rightarrow ADR$$

$$AD2 = (ADL \parallel_{\{c\}} ADR) \setminus \{c\}$$

- Tools such as **FDR** can be used to check model properties.
 - traditionally done at **design-time** for those using CSP approaches.

The CSP Process Algebra

- Developed by **Hoare**, 1978, 1985.
 - used to model and reason about interactions between parallel processes (arbitrary numbers).
 - fundamental interaction is **synchronisation** on **events**.
- Has both **operational** and **denotational** semantics, the latter providing for formal reasoning, including of process composition.
 - semantic models are **traces**, **failures** and **divergences**.
- For example:

$$ADL = in \rightarrow c \rightarrow ADL$$

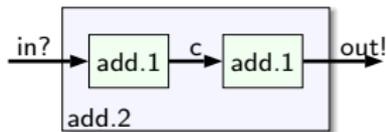
$$ADR = c \rightarrow out \rightarrow ADR$$

$$AD2 = (ADL \parallel_{\{c\}} ADR) \setminus \{c\}$$

- Tools such as **FDR** can be used to check model properties.
 - traditionally done at **design-time** for those using CSP approaches.

The CSP Process Algebra

- Developed by **Hoare**, 1978, 1985.
 - used to model and reason about interactions between processes (arbitrary numbers).
 - fundamental interaction is **synchronisation** on **events**.
- Has both **operational** and **denotational** semantics, the latter providing for formal reasoning, including of process composition.
 - semantic models are **traces**, **failures** and **divergences**.
- For example:



$$ADL = in \rightarrow c \rightarrow ADL$$

$$traces = \langle \rangle, \langle in \rangle, \langle in, c \rangle, \langle in, c, in \rangle, \dots$$

$$ADR = c \rightarrow out \rightarrow ADR$$

$$failures = (\langle \rangle, \{\}), (\langle in \rangle, \{in\}),$$

$$AD2 = (ADL \parallel_{\{c\}} ADR) \setminus \{c\}$$

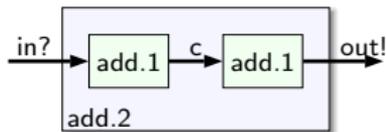
$$(\langle in, c \rangle, \{c\}), \dots$$

$$divergences = \{\}$$

- Tools such as **FDR** can be used to check model properties.
 - traditionally done at **design-time** for those using CSP approaches.

The CSP Process Algebra

- Developed by **Hoare**, 1978, 1985.
 - used to model and reason about interactions between processes (arbitrary numbers).
 - fundamental interaction is **synchronisation** on **events**.
- Has both **operational** and **denotational** semantics, the latter providing for formal reasoning, including of process composition.
 - semantic models are **traces**, **failures** and **divergences**.
- For example:



$$ADL = in \rightarrow c \rightarrow ADL$$

$$traces = \langle \rangle, \langle c \rangle, \langle c, out \rangle, \langle c, out, c \rangle, \dots$$

$$ADR = c \rightarrow out \rightarrow ADR$$

$$failures = (\langle \rangle, \{\}), (\langle c \rangle, \{c\}),$$

$$AD2 = (ADL \parallel_{\{c\}} ADR) \setminus \{c\}$$

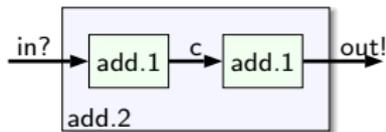
$$(\langle c, out \rangle, \{out\}), \dots$$

$$divergences = \{\}$$

- Tools such as **FDR** can be used to check model properties.
 - traditionally done at **design-time** for those using CSP approaches.

The CSP Process Algebra

- Developed by **Hoare**, 1978, 1985.
 - used to model and reason about interactions between processes (arbitrary numbers).
 - fundamental interaction is **synchronisation** on **events**.
- Has both **operational** and **denotational** semantics, the latter providing for formal reasoning, including of process composition.
 - semantic models are **traces**, **failures** and **divergences**.
- For example:



$$ADL = in \rightarrow c \rightarrow ADL$$

$$traces = \langle \rangle, \langle in \rangle, \langle in, out \rangle, \langle in, in, out \rangle, \dots$$

$$ADR = c \rightarrow out \rightarrow ADR$$

$$failures = (\langle \rangle, \{\}) , (\langle in, in \rangle, \{in\}) ,$$

$$AD2 = (ADL \parallel_{\{c\}} ADR) \setminus \{c\}$$

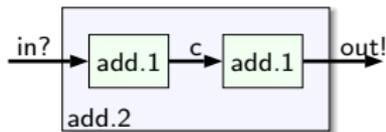
$$(\langle in, out \rangle, \{out\}) , \dots$$

$$divergences = \{\}$$

- Tools such as **FDR** can be used to check model properties.
 - traditionally done at **design-time** for those using CSP approaches.

The CSP Process Algebra

- Developed by **Hoare**, 1978, 1985.
 - used to model and reason about interactions between processes (arbitrary numbers).
 - fundamental interaction is **synchronisation** on **events**.
- Has both **operational** and **denotational** semantics, the latter providing for formal reasoning, including of process composition.
 - semantic models are **traces**, **failures** and **divergences**.
- For example:



$$\begin{array}{ll}
 \text{ADL} = in \rightarrow c \rightarrow \text{ADL} & \text{traces} = \langle \rangle, \langle in \rangle, \langle in, out \rangle, \langle in, in, out \rangle, \dots \\
 \text{ADR} = c \rightarrow out \rightarrow \text{ADR} & \text{failures} = (\langle \rangle, \{\}), (\langle in, in \rangle, \{in\}), \\
 \text{AD2} = (\text{ADL} \parallel \text{ADR}) \setminus \{c\} & (\langle in, out \rangle, \{out\}), \dots \\
 \{c\} & \text{divergences} = \{\}
 \end{array}$$

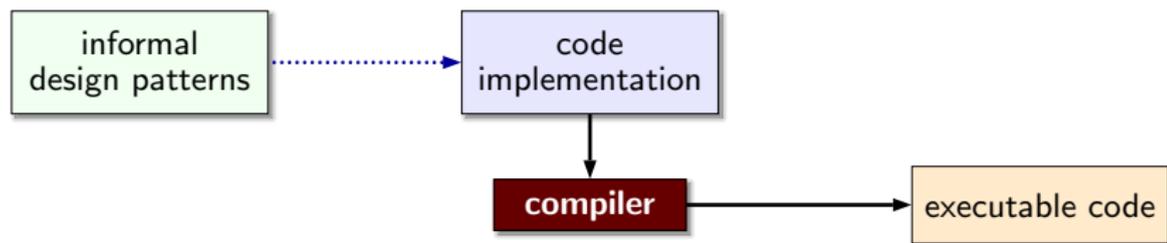
- Tools such as **FDR** can be used to check model properties.
 - traditionally done at **design-time** for those using CSP approaches.

Approach (mechanics)



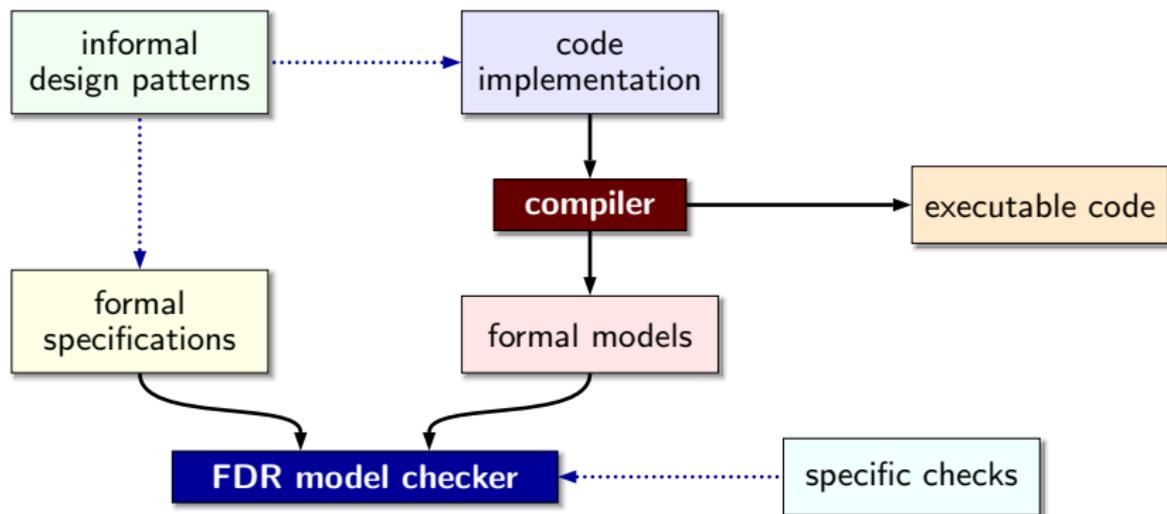
- Specifications can be created from occam- π model processes.
 - exemplar of **correct behaviour**, as well as a **model**.
- Generated models (in **XML** before conversion to **CSPm**) can be used in other checks (e.g. mobile escape analysis).

Approach (mechanics)



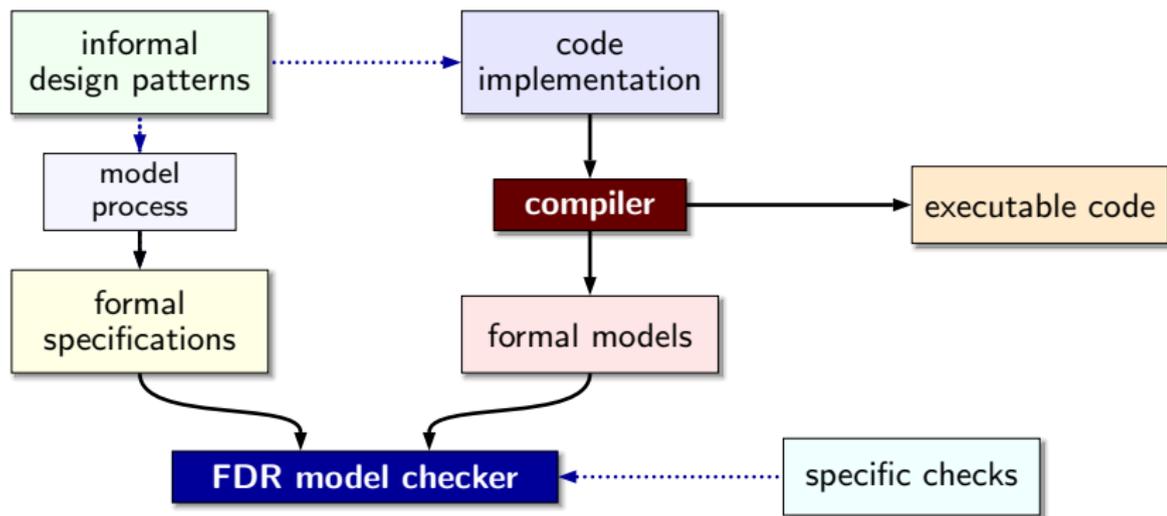
- Specifications can be created from occam- π model processes.
 - exemplar of **correct behaviour**, as well as a **model**.
- Generated models (in **XML** before conversion to **CSPm**) can be used in other checks (e.g. mobile escape analysis).

Approach (mechanics)



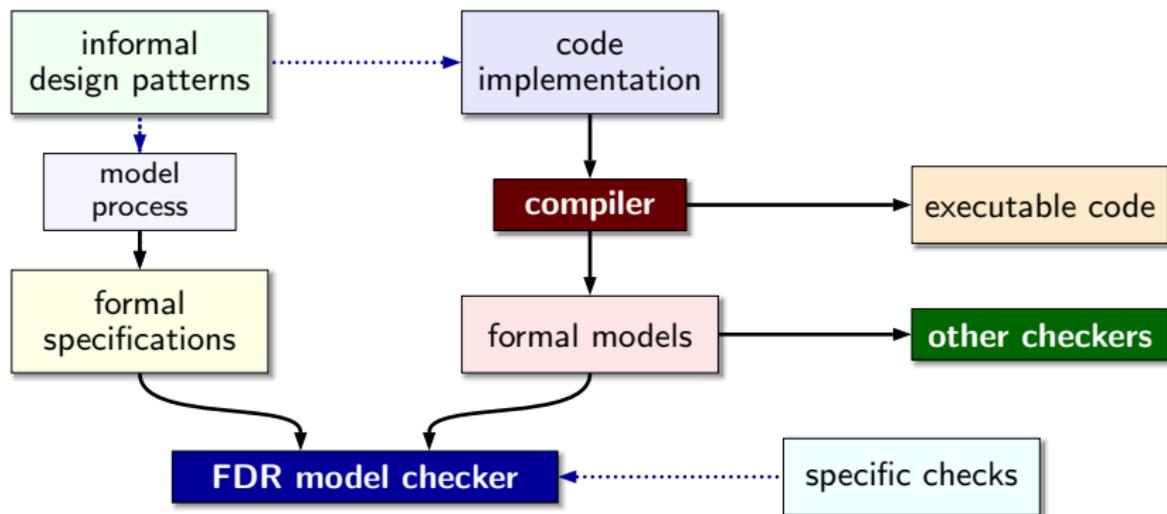
- Specifications can be created from `occam-π` model processes.
 - exemplar of **correct behaviour**, as well as a **model**.
- Generated models (in **XML** before conversion to **CSPm**) can be used in other checks (e.g. mobile escape analysis).

Approach (mechanics)



- Specifications can be created from occam- π model processes.
 - exemplar of **correct behaviour**, as well as a **model**.
- Generated models (in **XML** before conversion to **CSPm**) can be used in other checks (e.g. mobile escape analysis).

Approach (mechanics)



- Specifications can be created from $\text{occam-}\pi$ model processes.
 - exemplar of **correct behaviour**, as well as a **model**.
- Generated models (in **XML** before conversion to **CSPm**) can be used in other checks (e.g. mobile escape analysis).

Approach (practice)

- Have generated CSP models for several RMoX **device drivers**.
 - and checked *some* of these against specifications **successfully**.
- Some amount of **human** involvement required, particularly in deciding **what** to check.
- Checking **specific implementations** against **general specifications**:
 - **refinement** checks.
- Checking that the **composition** of model clients and servers is safe:
 - free from **deadlock** and **livelock** (divergence).
- Checking that **processes** behave correctly with respect to **mobile channel-types** (also refinement checks).

Approach (practice)

- Have generated CSP models for several RMoX **device drivers**.
 - and checked *some* of these against specifications **successfully**.
- Some amount of **human** involvement required, particularly in deciding **what** to check.
- Checking **specific implementations** against **general specifications**:
 - **refinement** checks.
- Checking that the **composition** of model clients and servers is safe:
 - free from **deadlock** and **livelock** (divergence).
- Checking that **processes** behave correctly with respect to **mobile channel-types** (also refinement checks).

Approach (simplified example)

- Starting with the earlier definitions of a **user-port** interface:
 - generate XML then CSPm models of the possible events:

```

PROTOCOL P.UPORT.IN
  CASE
    get.info
    set.dir; INT; INT
  :
PROTOCOL P.UPORT.OUT
  CASE
    info; INT
    error
    ok
  :
CHAN TYPE CT.UPORT
  MOBILE RECORD
    CHAN P.UPORT.IN in?:
    CHAN P.UPORT.OUT out!:
  :

```

```

<tagset name="CTPROT_CT_UPORT">
  <tag name="InPUportInGetInfo">
  <tag name="InPUportInSetDir">
  <tag name="OutPUportOutInfo">
  <tag name="OutPUportOutError">
  <tag name="OutPUportOutOk">
  <tag name="DoClaimCtUport">
  <tag name="DoReleaseCtUport">
</tagset>

```

```

datatype CTPROT_CT_UPORT = InPUportInGetInfo |
  InPUportInSetDir | OutPUportOutInfo |
  OutPUportOutError | OutPUportOutOk |
  DoClaimCtUport | DoReleaseCtUport

```

Approach (simplified example)

- 2 Define a **model** of the **server's** operation:
 - easily expressed in occam- π .

```

PROC servermodel (CT.UPORT? svr)
  INITIAL BOOL b.1 IS TRUE:
  WHILE TRUE
    svr[in] ? CASE
      get.info
      IF
        b.1
          svr[out] ! info; 0
        TRUE
          svr[out] ! error
      INT p, d:
        set.dir; p; d
      IF
        b.1
          svr[out] ! ok
        TRUE
          svr[out] ! error
  :

```



```

PSERVERMODEL_A0(svr) =
  (((svr.InPUportInGetInfo ->
    ((svr.OutPUportOutInfo -> SKIP) |~|
    (svr.OutPUportOutError -> SKIP))) []
  (svr.InPUportInSetDir ->
    ((svr.OutPUportOutOk -> SKIP) |~|
    (svr.OutPUportOutError -> SKIP)))));
  PSERVERMODEL_A0(svr)

PSERVERMODEL(svr) =
  PSERVERMODEL_A0(svr)

channel svr__0 : CTPROT_CT_UPORT

SYSTEM_PSERVERMODEL =
  PSERVERMODEL(svr__0)

```

Approach (simplified example)

- 3 Define a **model** of the (shared) **client's** operation:
- again, easily expressed in occam- π .

```

PROC sharedclientmodel (SHARED CT.UPORT? svr)
  INITIAL BOOL b.1 IS TRUE:
  INITIAL BOOL b.2 IS TRUE:
  WHILE b.2
    CLAIM cli!
    IF
      b.1
      SEQ
        cli[in] ! get.info
        cli[out] ? CASE
          INT inf:
            info; inf
            SKIP
          error
            SKIP
    TRUE
    SEQ
      cli[in] ! get.info
      cli[out] ? CASE
        error
          SKIP
        ok
          SKIP
  :

```



```

PSHAREDCLIENTMODEL_A0(cli) =
  (SKIP |~|
  ((cli.DoClaimCtUport -> ((cli.InPUportInGetInfo ->
  ((cli.OutPUportOutInfo -> SKIP) []
  (cli.OutPUportOutError -> SKIP))) |~|
  (cli.InPUportInSetDir ->
  ((cli.OutPUportOutError -> SKIP) []
  (cli.OutPUportOutOk -> SKIP)))));
  (cli.DoReleaseCtUport -> PSHAREDCLIENTMODEL_A0(cli)))

PSHAREDCLIENTMODEL(cli) =
  PSHAREDCLIENTMODEL_A0(cli)

channel cli__0 : CTPROT_CT_UPORT

SYSTEM_PSHAREDCLIENTMODEL =
  PSHAREDCLIENTMODEL(cli__0)

```

Approach (simplified example)

- 4 Starting with a **real** client implementation, generate formal model:
 - ... happens to be deliberately broken

```

PROC client (SHARED CT.UPORT! scli)
  INT pinfo:
  SEQ
    CLAIM scli!
    SEQ
      scli[in] ! get.info
      scli[out] ? CASE
        info; pinfo
      SKIP
  :

```



```

PCLIENT(scli) =
  (scli.DoClaimCtUport ->
   scli.InPUportInGetInfo ->
   scli.OutPUportOutInfo ->
   scli.DoReleaseCtUport ->
   SKIP)

```

Approach (simplified example)

- 5 Write a suitable CSPm wrapper to describe the particular **implementation** and **specification**:

```

PROC client (SHARED CT.UPORT! scli)
  INT pinfo:
  SEQ
    CLAIM scli!
    SEQ
      scli[in] ! get.info
      scli[out] ? CASE
        info; pinfo
      SKIP
  :

```

```

include "sharedclientmodel.csp"
include "client.csp"

channel c : CTPROT_CT_UPORT

MYSPEC = PSHAREDCLIENTMODEL(c)
MYIMPL = PCLIENT(c)

```



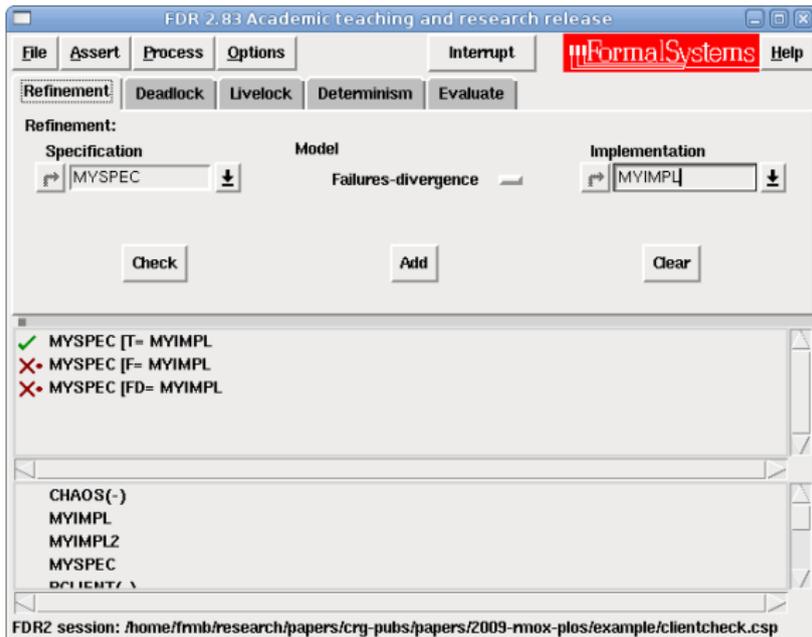
```

PCLIENT(scli) =
  (scli.DoClaimCtUport ->
   scli.InPUportInGetInfo ->
   scli.OutPUportOutInfo ->
   scli.DoReleaseCtUport ->
   SKIP)

```

Approach (simplified example)

- 6 Finally, ask FDR to perform refinement checks on these:
- for **traces**, **failures** and **divergences**.



```
include "sharedclientmodel.csp"
include "client.csp"
```

```
channel c : CTPROT_CT_UPORT
```

```
MYSPEC = PSHAREDCLIENTMODEL(c)
MYIMPL = PCLIENT(c)
```

Approach (simplified example)

- 6 Finally, ask FDR to perform refinement checks on these:
- for **traces**, **failures** and **divergences**.

The screenshot shows the FDR 2.83 Academic teaching and research release interface. The main window displays a refinement check for the specification `MYSPEC` against the implementation `MYIMPL`. The refinement type is set to `Failures-divergence`. The results show that the implementation fails to satisfy the specification in three ways:

- ✓ MYSPEC [T= MYIMPL
- ✗ MYSPEC [F= MYIMPL
- ✗ MYSPEC [FD= MYIMPL

The 'Accepts' dialog box is open, showing the observed and permitted traces for the implementation. The observed trace is `{c. OutPUPortOutInfo}` and the permitted trace is `{c. OutPUPortOutError, c. OutPUPortOutInfo}`.

The code snippets shown in the background are:

```
include "sharedclientmodel.csp"
include "client.csp"

channel c : CTPROT_CT_UPORT

MYSPEC = PSHAREDCLIENTMODEL(c)
```

The status bar at the bottom indicates the current session: `FDR2 session: /home/fmb/research/papers/cry-pubs/papers/2009-mox-plos/example/clientcheck.csp`

Limitations and Conclusions

- Limitations:
 - level of **abstraction** in models generated (arithmetic and other run-time errors).
 - size of models that are **practically** checkable in FDR2.
- Conclusions:
 - demonstrated that such checking is **possible**, and **one approach** for it — the formal underpinning of the language (CSP) is necessary.
 - gaining a **formal understanding** of how process-oriented systems behave (and how they may fail).
 - can be immediately applied to any occam- π systems (any process-oriented system in the long(er) term).
- Moreover, we can avoid expensive VM context switches in our OS implementation, leveraging **co-operative** concurrency for design and implementation, **because** we can **reason formally** about the behaviour of such systems.
 - thousands to millions of lightweight processes.
 - ~150ns context-switch / communication times across multiple cores.

Limitations and Conclusions

- Limitations:
 - level of **abstraction** in models generated (arithmetic and other run-time errors).
 - size of models that are **practically** checkable in FDR2.
- Conclusions:
 - demonstrated that such checking is **possible**, and **one approach** for it — the formal underpinning of the language (CSP) is necessary.
 - gaining a **formal understanding** of how process-oriented systems behave (and how they may fail).
 - can be immediately applied to any occam- π systems (any process-oriented system in the long(er) term).
- Moreover, we can avoid expensive VM context switches in our OS implementation, leveraging **co-operative** concurrency for design and implementation, **because** we can **reason formally** about the behaviour of such systems.
 - thousands to millions of lightweight processes.
 - ~150ns context-switch / communication times across multiple cores.

Limitations and Conclusions

- Limitations:
 - level of **abstraction** in models generated (arithmetic and other run-time errors).
 - size of models that are **practically** checkable in FDR2.
- Conclusions:
 - demonstrated that such checking is **possible**, and **one approach** for it — the formal underpinning of the language (CSP) is necessary.
 - gaining a **formal understanding** of how process-oriented systems behave (and how they may fail).
 - can be immediately applied to any occam- π systems (any process-oriented system in the long(er) term).
- Moreover, we can avoid expensive VM context switches in our OS implementation, leveraging **co-operative** concurrency for design and implementation, **because** we can **reason formally** about the behaviour of such systems.
 - thousands to millions of lightweight processes.
 - ~150ns context-switch / communication times across multiple cores.

The End

■ Questions..?

■ Acknowledgements:

- this work was funded by EPSRC grants EP/D061822/1 and EP/E049419/1.
- thanks to the anonymous reviewers for their feedback.

EPSRC Engineering and Physical Sciences
Research Council

RMoX



PLAS
Programming Languages and Systems

<http://rmox.net/>

<http://cosmos-research.org/>

<http://www.cs.kent.ac.uk/research/groups/plas/>