Adding Mobility to Networked Channel-Types

Mario Schweigler
Computing Laboratory, University of Kent
Canterbury, UK

KRoC.net – Overview

- Extension to KRoC
- Allows the distribution of occam-π channels (and channel-types) over networks
- Utilises new occam-π features to improve efficiency of implementation

Contents

- Components of KRoC.net
- Related extensions in occam-π
- Architecture and terminology
- Network-handles
- Network-channels and Network-channel-types (NCTs)
- Mobility of NCT-ends
- Conclusions and future work

Components of KRoC.net

- Consists of two essential parts:
  - KRoC.net kernel
  - Supportive code
- KRoC.net kernel implemented (largely) in occam-π itself
  - Allows efficient and secure implementation
- Supportive code to interface the KRoC.net kernel:
  - Compiler-generated code
  - Code in the occam-π kernel (KRoC runtime system)

Related Extensions

- Structural integrity support
- Mobile processes

Structural Integrity in occam-π

- Motivation:
  - New dynamic occam-π features have introduced hidden communication routes
  - Process header no longer clearly shows the interface of a process to its environment
    - Disadvantage compared to classical occam
    - A bit like in OO languages – but we want to do better!

Structural Integrity in occam-π

- New set of rules proposed (feedback welcome):
  - Definition:
    a) Channel-type-end parameters *may* be qualified as being **`GATE`** or **`HOLE`**. **`GATE`** and **`HOLE`** parameters are *live*.
    b) All other channel-type-ends are *dead* (i.e. locally declared channel-type-end-variables and parameters not qualified as **`GATE`** or **`HOLE`**).
  - Usage:
    c) A process may not communicate over a dead channel-type-end.

Structural Integrity in occam-π

- Assignment/Communication:
  - d) **GATE** channel-type-end parameters have **VAL** semantics – they may not be changed inside the process in whose header they are declared.
  - e) **GATE**, **HOLE** and dead channel-type-ends may be freely assigned/communicated to each other as long as this does not break Rule (d).
- Parameter-Passing:
  - f) Arguments for **GATE** parameters may only be live variables – unless the process is being **FORK**ed. If the process is being **FORK**ed, both live and dead arguments are allowed, as long as this does not break Rule (d).

Structural Integrity in occam-π

g) Inside the scope of a **CLAIM**, a claimed **SHARED** channel-type-end may be passed as an argument *only* to a non-shared **GATE** parameter of a process that is not being **FORK**ed.

h) **HOLE** parameters are initially undefined when a process starts. Arguments for **HOLE** parameters may be outer **HOLE** parameters of matching type which must be currently undefined, or the keyword **HOLE**. The latter may only be supplied to **FORK**ed processes. **HOLE** parameters have no return value (i.e. for the calling process they are still undefined when the callee process terminates).

i) Arguments for dead parameters may be dead or **HOLE** variables – unless the process is being **FORK**ed. If the process is being **FORK**ed, **GATE** variables are also allowed as arguments, as long as this does not break Rule (d).

Structural Integrity in occam-π

- Aim is that processes interact with environment only through formally declared live parameters
  - **GATE**s are fixed
  - **HOLE**s may be dynamically mapped to another channel-type-end by the process itself
  - External shape of the process does not change
- **FORK**ing may threaten structural integrity as well
  - It is proposed to 'label' **FORKING PROC**s so that the calling process is aware that a callee might fork another process

Structural Integrity in occam-π

- Impact on KRoC.net:
  - Important when moving NCT-ends
  - Since communication is only possible over live ends, there is no need to set up low-level network infrastructure when an NCT-end is moved into a dead variable
  - Thus, NCT-ends can be passed more efficiently between many nodes without the overhead of unnecessarily setting up low-level network infrastructure on intermediary nodes

Mobile Processes

- Important for KRoC.net:
  - The header of a **`MOBILE PROC`**ess may only contain ‘synchronisation objects’ like channels, barriers, etc., as well as **`GATE`** or **`HOLE`** channel-type-ends. It may *not* contain data items or dead channel-type-ends.
- Clean interface when moving mobile processes to remote nodes
  - Moving workspace of the process
  - Moving channel-type-ends that are stored in the workspace of the process
    - Channel-types stretched between nodes

Definitions

- A *node* is an occam- π program which is using KRoC.net, i.e. whose main process has declared a network-handle (see a few slides later).
- A *network-channel-type (NCT)* is a channel-type that connects several nodes, i.e. whose ends are on more than one node. A *network-channel* is the networked version of a 'classical' occam channel.

Definitions

- A group of nodes forms a logical *application*.
  - In the non-networked world, node and application would be congruent. In the networked world, an application is made up of several nodes.
    - Nodes can only communicate over NCTs that belong to the same application as they do
    - Each NCT can only connect nodes of the same application

Definitions

- A *Channel Name Server (CNS)* is an external server that administrates applications, nodes and NCTs.
  - Each application has a name that is unique within the CNS by which it is administrated.
  - Within the application, each node and each NCT has a unique name or automatically assigned ID. Each node has to register with a CNS before it can do any network communication.
  - NCTs are either allocated under an application-unique name *explicitly* via the CNS, or *implicitly* by moving ends of locally allocated channel-types to remote nodes.

Definitions

- The *network-type* is the type of a network infrastructure used by KRoC.net.
  - Every network-type has its own *KRoC.net kernel*. If a node declares a network-handle of a particular network-type, an instance of the respective KRoC.net kernel will be started.
  - TCP/IP currently only supported network-type
    - Adding support for others will be relatively easy, as the front-end of the KRoC.net kernel (which the supportive code interfaces) is the same for all network-types; just the back-end (the 'network driver') needs to be exchanged.

Network-Handles

- Built on the new concept of structural integrity
- To be able to allocate NCT-ends and network-channel-ends, a process must declare a *network-handle* as a **GATE** in its header
- Network-handle is the client-end of a channel-type defined in KRoC.net library (the server-end is held by the KRoC.net kernel):

```
CHAN TYPE NET.HANDLE
  MOBILE RECORD
    CHAN NET.HANDLE.REQ req?:
    CHAN NET.HANDLE.REPLY reply!:
:
```

Network-Handles

- Network-handles may be *typed*. If declared by the main process, they *must* be typed:

```
PROC main (CHAN BYTE keyb, scr, err,
           GATE NET.HANDLE$TCPIP! net)
  ... do stuff using 'net'
:
```

  - This will create supportive code that calls an instance of the TCP/IP version of the KRoC.net kernel
- Network-handles may be untyped if declared in a non-main process:

```
PROC my.proc (<...>, GATE NET.HANDLE! net, <...>)
  ... do stuff using 'net'
:
```

Network-Handles

- Network-handles may only be passed to network-handle parameters of the same type or to untyped network-handle parameters.
- Network-handles may be **`SHARED`**. The normal rules for **`SHARED`** channel-type-ends apply. If used to allocate NCT-ends etc. (see later), a **`SHARED`** network-handle must be **`CLAIM`**ed first
- Network-handles may only be declared in process headers and may not be communicated over channels

Network-Channels and NCTs

- Are the ‘backbone’ of KRoC.net
- Channel communication is the basic paradigm for distributed application development with KRoC.net
- Key issue is transparency – the channel communication over KRoC.net must work in the same way as the communication over local channels and channel-types

Joining an Application

- The first thing a node must do to be able to use KRoC.net
- Done by calling the following process:

```
PROC net.join.app (GATE NET.HANDLE! net,
                   VAL []BYTE cns, app.name, node.name,
                   RESULT INT result,
                   RESULT MOBILE []BYTE full.node.name)
```

- Implemented by a communication over the network handle
  - The supportive code communicates with the KRoC.net kernel who holds the server-end of the network-handle

Channel-Type-Blocks

- A *channel-type-block (CTB)* is the memory structure that defines a channel-type
  - Located in the dynamic mobilespace of the node; channel-type-end variables point to it
  - Local channel-types have one CTB
  - NCTs have one CTB per node, interconnected by KRoC.net

Channel-Type-Blocks

- To support NCTs, CTBs must be extended
- The CTB will be in one of the following states:
  - local
  - networked
  - localised

Channel-Type-Blocks

- When the CTB becomes networked (explicitly or implicitly – see later), it stores an *NCT-handle* (whose server-end is held by the KRoC.net kernel)
  - Similar to the network-handle, but for handling one particular NCT
- CTB gets *connected* when at least one live (i.e. **`GATE`** or **`HOLE`**) variable points to it
  - Only then will the the low-level network infrastructure be set up

Explicit Allocation of NCT-Ends

- Done by allocation processes such as:

```
PROC net.alloc.one2one.client (GATE NET.HANDLE! net,
         <CT>! the.cli, VAL []BYTE nct.name, RESULT INT result)
```

- This example connects the client-end of a one2one NCT (i.e. with a non-shared client-end and a non-shared server-end). Similar allocation processes for server-ends, and for any2one, one2any and any2any NCTs
- Structural integrity guaranteed by requiring a network-handle – if a process does not have one, it cannot allocate an NCT-end

Implementation of Network-Channels

- Network-channels are the networked version of single (classical occam) channels
- Approach similar to anonymous **`SHARED`** channels
- Each network-channel is implemented as an ‘anonymous’ NCT that contains exactly one channel

Implementation of Network-Channels

- The following declaration:

```
NET CHAN INT iw!:                -- These network-channel-ends
NET CHAN BOOL br?:               -- must be allocated before
SHARED NET CHAN BOOL sbw!:       -- we can communicate over them!
SHARED NET CHAN INT sir?:
... allocate iw!, br?, sbw!, sir?
... use iw!, br?, sbw!, sir?
```

Implementation of Network-Channels

- … would have the semantics of:

```
CHAN TYPE $anon.INT                -- compiler-generated type
  MOBILE RECORD
    CHAN INT x?:                   -- server-end holds reading-end
:

CHAN TYPE $anon.BOOL               -- compiler-generated type
  MOBILE RECORD
    CHAN BOOL x?:                  -- server-end holds reading-end
:

$anon.INT! iw$cli:
$anon.BOOL? br$svr:
SHARED $anon.BOOL! sbw$cli:
SHARED $anon.INT? sir$svr:
... allocate iw$cli, br$svr, sbw$cli, sir$svr
... use iw$cli, br$svr, sbw$cli, sir$svr
... resp. iw$cli[x], br$svr[x], sbw$cli[x], sir$svr[x]
```

Moving NCT-Ends

- When moving an end of a previously local channel-type to a remote node, the channel-type becomes networked
- It is allocated *implicitly*
  - Not by allocation the ends explicitly via a name …
  - … but implicitly by the supportive code when an end is sent away or when an end is received from a remote node

Moving NCT-Ends

- Three phase approach
- Example for sending the end of a previously local channel-type away:
  - **`DECODE.CHANNEL`** allocates the CTB implicitly; it gets registered with the CNS under a unique ID
  - **`DECODE.CHANNEL`** passes the pointer of the CTB to the KRoC.net kernel, who sends the NCT-end to the remote node; identified by its ID
  - The remote KRoC.net kernel passes the new end to **`ENCODE.CHANNEL`**, who also allocated the new CTB implicitly

Moving NCT-Ends

- Similar approach is taken when the end of an already networked channel-type is moved across the network
- In any case, the CTB of the received end will only be *connected* (i.e. the low-level network infrastructure be set up) when the end is stored in a live variable

Conclusions

- KRoC.net is now fully specified, the missing features for full transparency have been added
- Improvements have been made to make the use of KRoC.net as simple and intuitive as possible for the programmer
- Wherever possible, existing occam-π syntax has been used to specify KRoC.net

Conclusions

- Current state:
  - KRoC.net kernel almost completely implemented
  - Supportive code partly implemented
  - Feedback welcome on these plans, especially on the structural integrity proposals

Future Work

- Completion of the implementation of KRoC.net (kernel and supportive code)
- Comprehensive testing
- Performance benchmarks