

pony – The occam- π Network Environment

A Unified Model for Inter- and Intra-processor Concurrency

Mario Schweigler
Computing Laboratory, University of Kent
Canterbury, UK

Contents

- What is pony?
- Why do we need a unified concurrency model?
- Using pony
- Implementation of pony
- Conclusions and future work

CPA 2005: Mario Schweigler: pony Fringe Presentation

2

What is pony?

- Network environment for occam- π
(Name: anagram of [o]ccam, [p]i, [n]etwork, [y])
(Formerly known as 'KRoC.net')
- Allows a unified approach for inter- and intra-processor concurrency

CPA 2005: Mario Schweigler: pony Fringe Presentation

3

The Need for a Unified Concurrency Model

- It should be possible to distribute applications across several processors (or 'back' to a single processor) without having to change the components
- This transparency should not damage the performance
 - The underlying system should apply the overheads of networking only if needed in a concrete situation – this should be determined dynamically at runtime

CPA 2005: Mario Schweigler: pony Fringe Presentation

4

pony Applications

- A pony *application* consists of several *nodes*
- There are *master* and *slave* nodes – each application has one master node and any number of slave nodes
- Applications are administrated by an *Application Name Server (ANS)* which stores the network location of a master for a given application-name and allows slave nodes to 'look up' the master

CPA 2005: Mario Schweigler: pony Fringe Presentation

5

Network-channel-types

- The basic communication primitive between occam- π processes in pony are channel-types
- A *network-channel-type (NCT)* is the networked version of an occam- π channel-type
- It is transparent to the programmer whether a given channel-type is an intra-processor channel-type or an NCT

CPA 2005: Mario Schweigler: pony Fringe Presentation

6

Network-channel-types

- NCTs have the same semantics and usage as normal channel-types
 - Bundle of channels
 - Two ends, each of which may be shared
 - Ends are mobile
- Ends may reside on different nodes, and may be moved to other nodes

CPA 2005: Mario Schweigler: pony Fringe Presentation

7

Transparency in pony

- Semantic transparency
 - All *occam- π* **PROTOCOLS** can be communicated over NCTs
 - Semantics of communicated items are preserved, including **MOBILE** semantics
 - Handling of NCTs is transparent to the programmer
 - NCT-ends may be **shared**, like any other channel-type-end
 - NCT-ends are **mobile**, like any other channel-type-end, and can be communicated across distributed applications in the same way as between processes on the same node

CPA 2005: Mario Schweigler: pony Fringe Presentation

8

Transparency in pony

- Pragmatic transparency
 - pony infrastructure is set up dynamically when needed
 - If sender and receiver are on the same node, communication only involves 'traditional' channel-word access
 - If they are on different nodes, communication goes through the pony infrastructure (and the network)

CPA 2005: Mario Schweigler: pony Fringe Presentation

9

NCTs and CTBs

- A *network-channel-type* (NCT) is a logical construct that comprises an entire networked channel-type
 - Has a unique ID (and a unique name if allocated explicitly) across a pony application
- A *channel-type-block* (CTB) is the memory block of a channel-type on an individual node
 - Either the only CTB of an intra-processor channel-type
 - Or one of possibly many CTBs of an NCT

CPA 2005: Mario Schweigler: pony Fringe Presentation

10

Using pony

- There are a number of public processes for:
 - Starting pony
 - Explicit allocation of NCT-ends
 - Shutting down pony
 - Error-handling
 - Message-handling

CPA 2005: Mario Schweigler: pony Fringe Presentation

11

Starting pony

- Startup processes:
`pony.startup.(u|s)nh[(u|s)eh[.iep]][.mh]`
- Returns:
 - A (possibly shared) network-handle
 - Used for allocation and shutdown
 - A (possibly shared) error-handle if required
 - Used for error-handling
 - A message-handle if required
 - Used for message-handling

CPA 2005: Mario Schweigler: pony Fringe Presentation

12

Starting pony

- Startup process contacts the ANS
- If our node is the master, its location is stored by the ANS
- If our node is a slave
 - Startup process gets location of master from ANS
 - Connects to master
- On success, startup process starts *pony kernel* and returns the requested handles
- Otherwise returns error

CPA 2005: Mario Schweigler: pony Fringe Presentation

13

Explicit Allocation of NCT-ends

- Allocation processes:
`pony.alloc.(u|s)(c|s)`
- The ends of an NCT may be allocated on different nodes at any given time
- Allocation process communicates with pony kernel via network-handle
- Versions for unshared/shared client-ends/server-ends
- An *explicitly* allocated NCT is identified by a unique NCT-name stored by the master node

CPA 2005: Mario Schweigler: pony Fringe Presentation

14

Explicit Allocation of NCT-ends

- If parameters are valid, allocation process allocates and returns the NCT-end
- Allocation process returns error if there is a mismatch with previously allocated ends of the same name, regarding:
 - Type of the channel-type
 - Shared/unshared properties of its ends

CPA 2005: Mario Schweigler: pony Fringe Presentation

15

Implicit Allocation by Moving

- Any channel-type-end, including NCT-ends, may be moved along a channel
- If an end of a locally declared channel-type is moved to another node (along a channel of an NCT) for the first time, this channel-type is *implicitly* allocated by the pony kernel
 - That channel-type automatically becomes an NCT
 - Programmer does not need to take any action himself
 - Does not even need to be aware whether the end is sent to a process on the same or on a remote node

CPA 2005: Mario Schweigler: pony Fringe Presentation

16

Implicit Allocation by Moving

- If an end of an NCT arrives at a node for the first time, a new CTB for that NCT is allocated on the receiving node
 - Again, this is transparent for the occam- π programmer – the process receiving the end does not need to be aware whether it is coming from a process on the same node or on a remote node

CPA 2005: Mario Schweigler: pony Fringe Presentation

17

Shutting down pony

- Shutdown process:
`pony.shutdown`
- Shutdown process communicates with pony kernel via network-handle
- Must be called after all activity on (possibly) networked channel-types has ceased
- If node is master, pony kernel notifies the ANS about the shutdown
- pony kernel shuts down its internal components

CPA 2005: Mario Schweigler: pony Fringe Presentation

18

Error-handling

- Used for the *detection* of networking errors during the operation of pony applications ('logical' errors are handled by suitable results returned by the public pony processes, see above)
- Not transparent – because there is currently no built-in error-handling or fault-tolerance mechanism in occam- π on which pony's error-handling could be built

CPA 2005: Mario Schweigler: pony Fringe Presentation

19

Error-handling

- Programmer can:
 - Either do it the 'traditional' occam- π way, namely trust that there will be no errors and live with the consequences otherwise
 - Or use pony's (non-transparent) error-handling mechanism to detect possible networking errors

CPA 2005: Mario Schweigler: pony Fringe Presentation

20

Error-handling

- If an error-handle was requested from the startup process, the pony kernel has started an error-handler
- Error-handler stores networking errors that occur
- Error-handling processes:
`pony.err.*`
- They use the error-handle to communicate with the error-handler

CPA 2005: Mario Schweigler: pony Fringe Presentation

21

Error-handling

- Error-handling mechanism:
 - Set error-point before an operation (an initial error-point may be returned by the startup process)
 - Do operation, wait for it to complete (maybe set a timeout – this is up to the programmer and not part of pony)
 - If needed (maybe after a timeout has expired), check for errors that happened after a given error-point

CPA 2005: Mario Schweigler: pony Fringe Presentation

22

Error-handling

- pony errors are records containing information about the type of error, as well as about the remote node involved
- Programmer can find out about the current remote node of a given NCT and only check for errors that involve that particular node

CPA 2005: Mario Schweigler: pony Fringe Presentation

23

Message-handling

- Used for outputting messages from the pony kernel
 - Status messages
 - Error messages
- If a message-handle was requested from the startup process, the pony kernel has started a message-handler
- Message-handler stores messages from the pony kernel

CPA 2005: Mario Schweigler: pony Fringe Presentation

24

Message-handling

- Message-outputters:
`pony.msg.out.(u|s)(o[(u|s)e]|e)`
- They use the message-handle to communicate with the message-handler
- Run in parallel with everything else
- Status messages are outputted to a (possibly shared) output channel (typically `stdout`)
- Error messages are outputted to a (possibly shared) error channel (typically `stderr`)

CPA 2005: Mario Schweigler: pony Fringe Presentation

25

Configuration

- Done via simple configuration files
- Used to configure
 - Network location of a node
 - Network location of the ANS
- All settings may be omitted
 - In this case either defaults are used or the correct setting is detected automatically

CPA 2005: Mario Schweigler: pony Fringe Presentation

26

Implementation of pony

- Internal components of pony:
 - Main kernel
 - Protocol-converters
 - Decode-handler and encode-handler
 - CTB-handler and CTB-manager
 - NCT-handler and NCT-manager
 - Link-handler and link-manager
 - Error-handler
 - Message-handler

CPA 2005: Mario Schweigler: pony Fringe Presentation

27

Main pony Kernel

- Interface between the public processes and the internal pony components
- Deals with:
 - Explicit allocation of NCT-ends
 - Shutdown

CPA 2005: Mario Schweigler: pony Fringe Presentation

28

Protocol-converters

- Protocol-decoder and protocol-encoder
- Implemented in C using the `occam-π` C interface (CIF)
- Interface between the `occam-π` world and pony
- Service one individual channel of a CTB
- Provide **PROTOCOL** transparency by decoding any given `occam-π` **PROTOCOL** to a format used internally by pony, and reverse

CPA 2005: Mario Schweigler: pony Fringe Presentation

29

Decode-handler and Encode-handler

- Interface between the protocol-converters and the CTB-handler
- Deal with:
 - Efficient handling of networked communications over a channel of an NCT
 - Movement of NCT-ends
 - Implicit allocation of NCT-ends if applicable (see above)

CPA 2005: Mario Schweigler: pony Fringe Presentation

30

CTB-handler and CTB-manager

- CTB-handler deals with:
 - Claiming and releasing ends of an individual CTB
 - Communication over its channels
- CTB-manager
 - Manages CTB-handlers on a node
 - Creates new ones if a new networked CTB appears on a node (explicitly or implicitly)

CPA 2005: Mario Schweigler: pony Fringe Presentation

31

NCT-handler and NCT-manager

- Reside on master node
- NCT-handler
 - Deals with claiming and releasing NCT-ends
 - Keeps queues of nodes that have claimed the client-end or the server-end of an NCT
 - Equivalent of the client- and server-semaphores of an intra-processor CTB
- NCT-manager
 - Manages NCT-handlers
 - Creates new ones if a new NCT is allocated (explicitly or implicitly)

CPA 2005: Mario Schweigler: pony Fringe Presentation

32

Link-handler and Link-manager

- Link-handler
 - Handles a link between two nodes
 - All network communication between these two nodes is multiplexed over that link
- Link-manager
 - Manages link-handlers
 - Creates new ones if a new link is established to another node

CPA 2005: Mario Schweigler: pony Fringe Presentation

33

Modular Design of pony

- Link-handler, link-manager and ANS are specific for a particular network-type
 - Currently supported: TCP/IP
- All other components of pony are not network-type-specific
- Supporting other network-types only requires rewriting the network-type-specific components
 - They communicate with the non-network-type-specific components of pony via the same interface

CPA 2005: Mario Schweigler: pony Fringe Presentation

34

Error-handler and Message-handler

- Error-handler
 - Stores networking errors
 - Interface between internal pony components and public error-handling processes
- Message-handler
 - Stores status and error messages
 - Interface between internal pony components and message-outputters

CPA 2005: Mario Schweigler: pony Fringe Presentation

35

Conclusions

- pony and occam- π provide a unified concurrency model for inter- and intra-processor applications
 - Achieving semantic and pragmatic transparency
- Simple handling of pony for the occam- π programmer
 - Minimum number of public processes for basic operations (startup, explicit allocation, etc.)
 - Runtime operation handled automatically and transparently by the pony kernel
 - Simple (mostly automatic) configuration

CPA 2005: Mario Schweigler: pony Fringe Presentation

36

Future Work

- Adding support for new `occam-π` features (which are added to `occam-π` currently or in the future)
 - Mobile processes
 - Mobile barriers
 - Enhancing pony's error-handling (making it transparent) by supporting proposed built-in `occam-π` fault-tolerance mechanisms

CPA 2005: Mario Schweigler: pony Fringe Presentation

37

Future Work

- Enhancing performance
 - Enhancing network latency by supporting proposed `occam-π` features such as **BUFFERED** channels and 'behaviour patterns' (for ping-pong style communication)
 - Supporting the proposed **GATE/HOLE** mechanism in `occam-π` by not setting up the internal pony components for a networked CTB until one of its ends is used for communication (i.e. not if an NCT-end is just 'passing through' a node)

CPA 2005: Mario Schweigler: pony Fringe Presentation

38

Future Work

- Enhancing performance
 - Adding a new **LOCAL** keyword to `occam-π` to allow the declaration of non-networkable channel-type-ends without the overheads that are necessary for networkable ones
- Integrating pony into RMoX
 - Due to pony's modular design, only the network-type-specific components need to be adapted

CPA 2005: Mario Schweigler: pony Fringe Presentation

39

Future Work

- Other things
 - Adding support for networked 'plain' channels – fairly straightforward, using anonymous channel-types similar to the **SHARED CHAN** approach
 - Support for platforms with different endianness
 - Support for security, encryption
 - Support for launching nodes automatically (e.g. by using Grid-style cluster management)
 - Support for Zeroconf-style setup on local clusters (i.e. without the need for the ANS)

CPA 2005: Mario Schweigler: pony Fringe Presentation

40