An Investigation into Distributed Channel Mobility Support for Communicating Process Architectures

Kevin CHALMERS and Jon KERRIDGE
School of Computing, Edinburgh Napier University, Edinburgh, EH10 5DT
{k.chalmers, j.kerridge}@napier.ac.uk

Abstract. Localised mobile channel support is now a feature of Communicating Process Architecture (CPA) based frameworks, from JCSP and C++CSP to occam-π. Distributed mobile channel support has also been attempted in JCSP Networking and occam-π via the pony framework, although the capabilities of these two separate approaches is limited and has not led to the widespread usage of distributed mobile channel primitives. In this paper, an initial investigation into possible models that can support distributed channel mobility are presented and analysed for features such as transmission time, robustness and reachability. The goal of this work is to discover a set of models which can be used for channel mobility and also supported within the single unified protocol for distributed CPA frameworks. From the analysis presented in this paper, it has been determined that there are models which can be implemented to support channel end mobility within a single unified protocol which provide suitable capabilities for certain application scenarios.

Keywords. mobile channels, distributed computing, protocol support.

Introduction

Recent work in Communicating Sequential Processes for Java (JCSP) Networking has focused on refining the underlying architecture and protocol, as well as providing support for distributed mobility of processes and channels. Last year [1], a universal protocol to support distributed operations across all CPA frameworks was introduced. The initial version of the protocol was designed to reduce resource usage within JCSP Networking, as well as promote interoperability between the other CPA frameworks by having a well defined set of primitive network messages that can be understood by languages as diverse as occam-π and Python. The next stage in this work is to also provide channel end mobility support in the protocol such that channel ends can be passed between, for example, a JCSP application and an occam-π application. The work presented in this article is an initial investigation into models to support distributed channel mobility within the CPA network protocol. This will lead to dynamic topology support, which is useful in fields such as mobile agents [2], complex systems [3] and pervasive computing [4]

The rest of this paper is broken down as follows. In Section 1, a discussion on distributed mobility in CPAs is presented, looking at the requirements to support such functionality. Section 2 presents potential models to support distributed channel mobility, and Section 3 analyses certain attributes of these models. Section 4 discusses possible protocol integration for these models. Section 5 presents future work and Section 6 provides conclusions.

1. Distributed Mobility in CPAs

Distributed mobility in CPAs refers to the ability to migrate a process or channel end in a distributed CPA application from one network node to another in a manner that is transparent to the application (this is referred to as logical mobility [2]). Localised mobility support has been possible in JCSP since the initial version due to the pass-by-reference semantics of Java. Mobility support for occam was introduced with occam-π [5], the emphasis being on providing correct mobility support. Distributed mobile processes have also been implemented in both JCSP [6] and occam-π [3], the former having further support for code mobility. Distributed channel end mobility has also been implemented in JCSP [6] and the pony framework supported distributed channel mobility for occam-π [7]. Trap [3] is a successor to pony that currently has no support for channel mobility. There are difficulties in implementing distributed channel and process mobility in a manner that still emits the behaviour that we would expect from both localized and distributed mobility.

1.1 Difficulties with Distributed Mobility against Localised Mobility

Previous work examining the challenges of mobility in CPA frameworks was highlighted in [6], and is summarized in Table 1:

Table 1. Complexity of mobility.

Mobile Primitive	**Local Mobility**	**Distributed Mobility**
Input Channel End	Simple	Difficult
Output Channel End	Simple	Simple
Simple Process	Simple	Moderate
Complex Process	Simple	Very Difficult

On a single machine, mobility of channel ends and processes is relatively simple, requiring the passing of a reference from one process to another, occam-π hiding this underlying transaction from the developer. For distributed mobility, the implementation is more difficult. Output Channel End mobility is relatively simple as it normally only requires the transmission of an address to send messages to. Input Channel End mobility is difficult as it requires informing any Output Channel End(s) connected to the Input Channel End. Simple Process mobility refers to single processes, and the moderate difficulty refers to the inclusion of a code mobility system to support transparent process mobility. Complex Process mobility requires the suspension and subsequent resumption of a process network which has internal communication between the migrating processes.

As the table indicates, the difficult problems to solve are Input Channel End mobility and Complex Process Mobility. Output Channel End mobility is solved based on the chosen Input Channel End mobility solution, and Simple Process mobility has been solved in JCSP via code mobility support [6]. The focus of this article is Input Channel End mobility, which helps enable Complex Process mobility as discussed in Section 1.3.

1.2 Code Mobility

Logical mobility is discussed within the context of code mobility [8]. The code mobility paradigm discusses various models of mobile software components (e.g. mobile agents and client-server). Code mobility is also categorised into strong and weak mobility, the difference lying in the movement of active or passive components. An active component is one that has its own thread of control, whereas a passive component does not. Weak code

mobility requires non-stateful movement of a component from one networked node to another. Strong code mobility requires capturing the current execution state of an active component and transferring this to the new location. Both approaches include passive state capture (e.g. attributes of an object) and mobility of code. Execution state can be considered as the instruction pointer and call stack of an individual thread that is to be transferred.

Strong code mobility is related to complex process mobility as discussed in Section 1.1. The difficulty in a platform such as Java is that the application developer does not have access to the internal instruction pointer or call stack of a thread, and therefore state capture of active components is difficult. Attempts have been made to overcome this limitation (for example see [9,10,11]), although they require modified Java Virtual Machines (JVMs) or compilers.

The code mobility viewpoint of logical mobility has limitations when analysed within software architecture models, as shall be discussed in the following two sub-sections.

1.2.1 Software Architecture

Generally, software architectures are defined by components and the connections between the components. For example, with CPA there are process components and channel connectors, and for object-orientation there are objects and the references between them. A system can be defined architecturally by the set of components and the connection relationships between them.

Architectural elements can be further analysed by defining the ports (the inputs and outputs of a component) and the connection ends (inputs to a connection and the outputs from it). In CPAs, connection ends correspond to channel ends, although these are classified from the process point of view. Therefore a channel output end is the output from a process into a channel, and not the output from a channel. Ports can be considered as the set of events which a process operates on. This definition is illustrated in Figure 1.

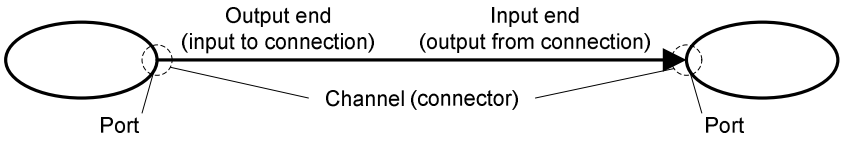

Figure 1. CPA architecture.

1.2.2 Limitations of the Code Mobility View

Code mobility has a limitation from a software architecture point of view in that connection mobility is not considered. This leads to the situation where a mobile component in a code mobility system can be viewed as an isolated piece of data, an isolated component (which may have internal components) or a whole application with all the internal components and connectors persisted. A component does not take its external connections with it when it migrates. Initial work on the π-Calculus [12] considered that process mobility was enabled by channel mobility, whereas code mobility has not considered this approach in depth.

There has been some discussion on coordination mobility support within logical mobility. Roman et al. [13] has argued that coordination and location are the most important factors for logical mobility as coordination mobility enables the decoupling of components. Roman also argues that coordination mobility should be considered separate to component mobility. Phillips et al. [14] has argued for better modelling of communication between mobile components. Therefore, the current focus on distributed CPA mobility is on connection mobility to support component mobility.

1.3 Component Mobility

We define a more concise model of component mobility which overcomes the limitations of the code mobility model. A mobile element in a code mobility system can be considered to have the following structure:

- *Code* – the code defining the structure and behaviour of the mobile element. This is required in a code mobility system.
- *State* – the current state of the mobile element. This is further categorised into:
  - *Passive state* – the data attributes of the mobile component. This is required in a code mobility system.
  - *Active state* – the execution state of the mobile element. For a strong code mobility system this is required.

Our view of a mobile component has the following structure:

- *Type* – the type of the component. This describes its structure and behaviour. The type is required for interpretation at the receiving node in a distributed application. Further, the type may also include:
  - *Code* – the code, which may have to be loaded at the receiving end to allow interpretation of the mobile component. This is not a requirement for component mobility, particularly if we want to allow component mobility between diverse frameworks.
- *State* – the current state of the mobile component. This has three sub-elements:
  - *Connection state* – any connections to external components that the mobile component may have. This is a requirement for strong component mobility.
  - *Data state* – the attributes of the mobile component. This is required for any mobile component.
  - *Behaviour state* – the current execution state of the component. This is a requirement for strong code mobility.

Our model of component mobility allows for full definition of any mobile element that a system may have. For example, as only the type and data state are required for the most primitive form of mobile component, we can define mobile data (a simple message) within the mobile component structure.

1.4 Comparing Component Mobility to Code Mobility

In code mobility, strong and weak mobility is distinguished by the capturing and sending of current execution state with the mobile element. Component mobility requires both connection state and behaviour state to determine strong mobility.

Unlike code mobility, there is no express requirement in component mobility to transfer code with the mobile element. The reason to take this view is twofold. Firstly, we wish to be able to map primitive (well known) data messages within our definition. For example, 32-bit integers and strings are standard data types with no functionality (code) associated with them. Secondly, we want to acknowledge the ability to send a mobile component from one framework to another. It might become feasible to have strong component mobility from a JCSP application to an occam-π application. No cyclic references could be within the sent message. Having a uniform method of connection mobility between frameworks is required to support inter-framework component mobility.

The main addition that component mobility brings is the inclusion of connection state. This is not the internal connectivity of the mobile component but the external connected interface. Retaining this state allows the migration of the component in a manner that is transparent to other components in the system as communication between components remains intact. Adequate connection state migration therefore enables transparent component mobility. With CPAs, channels are treated as first class, thereby decoupling a component from its connections. This is important to enable strong mobility of component and connection.

1.5 Difficulties in First Class Mobility of Object-Oriented Applications

Object-orientation does not exhibit both first class component and connection mobility. When running on a single machine, an object-oriented application passes references to objects during method invocation, and thus only connection mobility is evident. For a distributed application, the reverse is evident with an object being serialized and copied from one networked node to another. There is no concept of passing an object reference from one application to another. There is a definite machine boundary in an object-oriented application which separates the distributed from the localised.

Because of the limitation of object-orientation, mobility support in CPA can lead to more transparent mobile applications. The following section describes seven different models that can support distributed channel end mobility, and Section 3 analyses some of the properties of these models. A more in depth discussion is provided elsewhere [15].

2. Models of Distributed Connection Mobility

Through examination of other techniques to support connection mobility, seven possible models to support channel mobility have been discovered. These models are described in the following sections. This is not an exhaustive collection of models, although we have surveyed available work within reason.

2.1 One-to-One Networked Channel

Networked channels are Any-to-One in that any number of output ends may connect to an input end. As it is unknown how many output ends may be connected to an input end, informing output ends of the movement of an input end is not a one-to-one communication. The One-to-One model is illustrated in Figure 2.

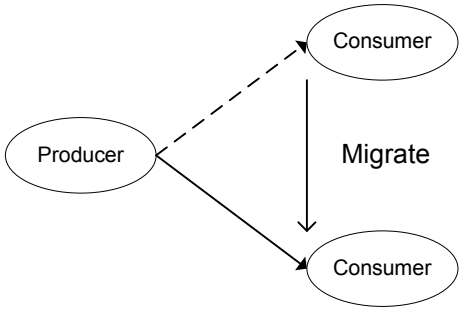

Figure 2. One-to-One networked channel.

Muller [16] has presented a mobile channel protocol that supports One-to-One communication. Channel end states are used and vary based on whether the end is locally or remotely connected, and each channel end knows the location of its corresponding

partner. When a channel end migrates, it informs its companion of the new location once it has arrived. Mobility is easier in comparison to the standard Any-to-One model as it can be guaranteed that the companion channel end has been notified of the new location.

2.2 Name Server

Mobile channel locations contained on a server is the approach taken by pony [7, 17]. Each channel is allocated an identifier unique to the application context (the set of networked nodes that make up a single pony application). Identifiers are managed by a server which tracks the current location of the channel. When the channel end is migrated the location is updated on the server. An output end connected to an input end can resolve this location, and then connect directly to the input end. If the input end should later move the output end retrieves the new location from the central server. This model is basically an extension of the common broker architecture used in distributed systems, and is illustrated in Figure 3.

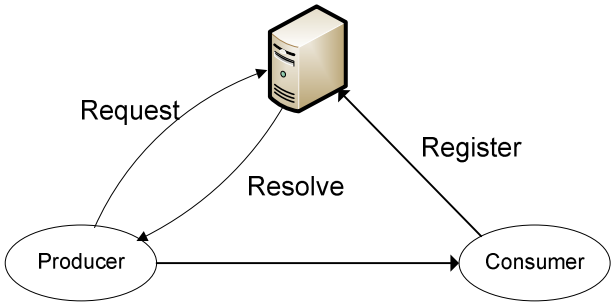

Figure 3. Name server.

All the other models may use a name server for channel end resolution, although this is not a requirement. A channel can be connected using only the address. This model requires a name server, and also adds functionality to the server to support channel end mobility.

2.3 Message Box

Message boxes are the approach used within mobile agent frameworks [18], and the model previously proposed for JCSP Networking channel mobility [6]. The node declaring the input channel end creates a message box process, which allows the output end to send to a static address, and the input channel end to request the next message from the message box. The message box model is illustrated in Figure 4.

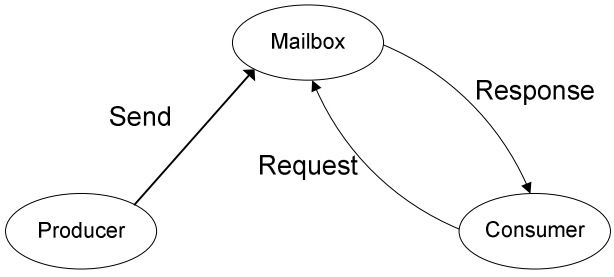

Figure 4. Message box.

2.4 Message Box Server

The message box model can be combined with a server allowing creation of message boxes on the server instead of locally on a node [19]. Apart from the requirement of server creation, the operation of the server controlled message box is identical to the standard message box model. This model is illustrated in Figure 5.

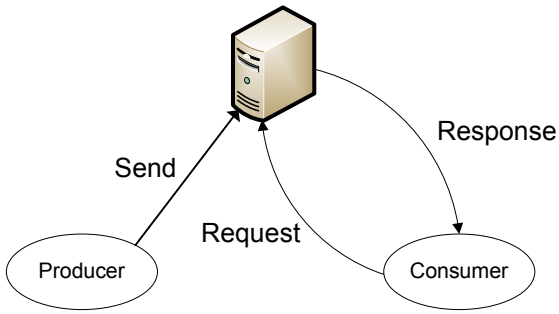

Figure 5. Message box server.

2.5 Chain

The chain model [20] requires each previous location of a channel end to forward any message onto the next location until the message reaches the current location of the input end. When an input end arrives at a new location it informs the previous location of the new location. When an output end moves, the previous location is sent with the migration message, which is used to send to the previous output end location. Thus a chain of connections is formed, and any message must traverse the entire length of the chain. The model is illustrated in Figure 6.

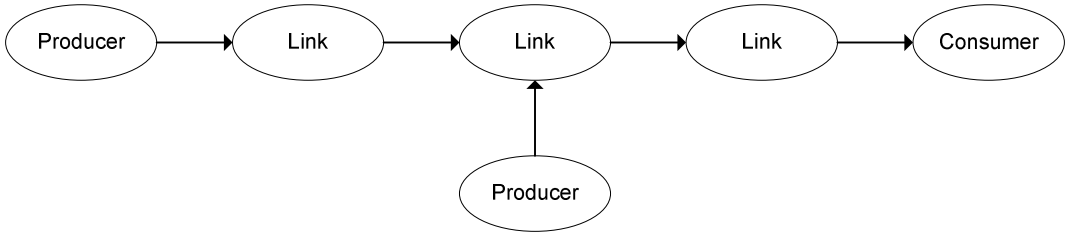

Figure 6. Chain.

As networked channels are Any-to-One, there will be chains of various lengths in operation. The length from the original input location to the current input location is always determined by the number of migrations that have been made by the input end. The length of the output end(s) depends on how far the output end has moved from the original location. Thus, as different output ends may traverse different distances, there will be multiple chain lengths in operation.

2.6 Reconfiguring Chain

To overcome the loop and transmission problems of the chain model [21], the chain can reconfigure itself by finding shortcuts to a previous link. Any loop is therefore removed and transmission time may become reduced whenever the chain is shortened. The reconfiguring chain model is illustrated in Figure 7.

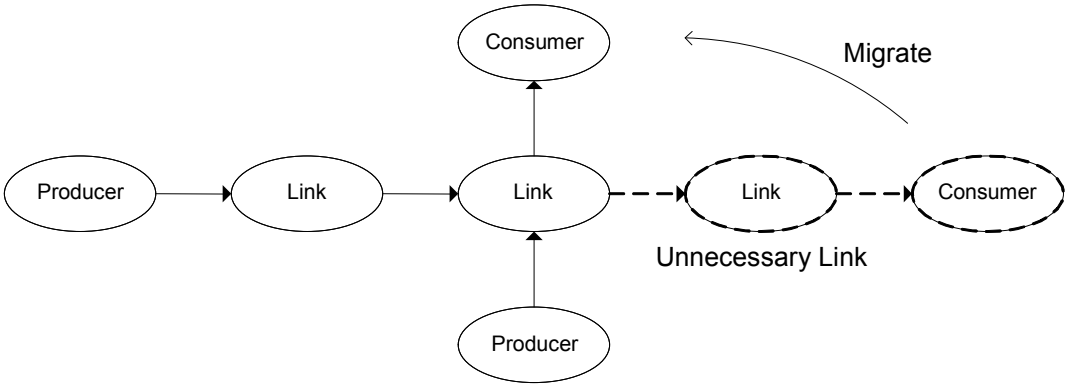

Figure 7. Reconfiguring chain.

To achieve reconfiguration, a migrating channel end takes all previous location in the chain. On arrival, the locations are iterated through and reconnection is attempted to the oldest possible link in the chain. Loops are removed as a node can always shortcut to itself. Transmission time for messages can be reduced as the most direct route between two nodes is used instead of the total distance travelled by the mobile end.

2.7 Mobile IP

Mobile IP [22] is used for physical device mobility within IP based networks. Connections are registered with a home agent responsible for forwarding messages onto the current location of the input end. When a connection migrates, it informs the home agent, which buffers messages until the new location is resolved. The new location address is generated by the foreign agent within the domain of the channel end's new location. The home agent forwards received messages to the foreign agent, which forwards messages to the channel end's new location. Whenever the mobile end moves, the foreign agent informs the home agent, and the same migration process occurs. This model is illustrated in Figure 8.

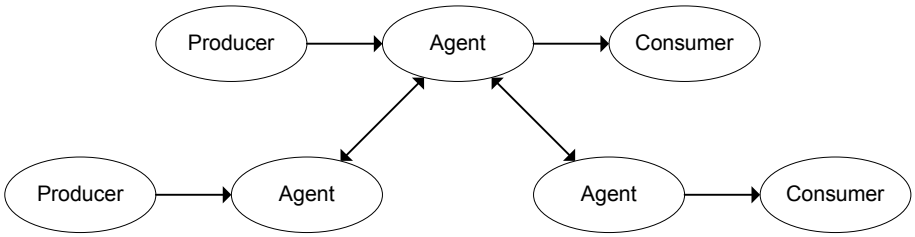

Figure 8. Mobile IP.

To enable mobility between network sub-domains, tunnelling is used to allow messages to be sent to the new foreign agent. Tunnelling can be reproduced in a mobile channel context by utilizing a chain of agents that forward messages to the respective channel end location or next agent. The difference between an agent chain and a normal chain is that the agent chain is a fixed architecture which only grows when contact with a new domain occurs. This creates a hybrid model of chaining, server and message box. The agents act as both gateways between domains and routers of messages.

2.8 Advantages and Disadvantages

Each of these models has certain advantages and disadvantage in comparison to the other models. These advantages and disadvantages are summarized in Table 2. These advantages and disadvantages are of interest as they highlight where some of the models are more suitable than others in certain application scenarios.

Table 2. Advantages and disadvantages of mobility models.

Model	**Advantages**	**Disadvantages**
One-to-One	Direct connection; simple model	No support for Any-to-One connections.
Name server	Direct connection	Requires a name server.
Message box	All transmissions require only one hop	Requires origin node to host the message box.
Message box server	All transmissions require only one hop	Requires a server to host the message box; server may become overloaded.
Chain	Channel ends can travel freely	Requires all previous nodes to support the chain; transmission time increases with each migration; single node failure can break multiple chains; loops may exist in the chain.
Reconfiguring chain	Channel ends can travel freely	Reconfiguring the chain takes time; some of the chain disadvantages may still exist.
Mobile IP	Channel ends can travel freely	Requires a backbone of agents to support mobility; loops may exist.

3. Analysis of Connection Mobility Models

For analysis of the different models, a restricted addressing layout of standard TCP/IP based communication networks is used. A network domain may consist of several sub-domains, which in turn consist of sub-domains, etc. At the root of the domain tree is the global domain. Each node in the tree can be allocated an identifier to represent the domain in the hierarchy that it belongs to. Messages are sent between members of domains. Figure 9 presents an example domain tree. This layout is not a representation of physical network layout, but rather the logical domain addressing mechanism in place.

Each node in the tree has an identifier based on its domain branch. For example, leaf E has identifier G.A.C.E. A simplistic viewpoint of connectivity is taken in that members of a sub-domain may connect to members of the same sub-domain and members of parent domains. Thus, any leaf in the tree can connect to any domain further up its branch until the global domain root node is reached. For example, a member of G.A.C.E. can connect to a member of G.A.C., G.A., and G. This form of connectivity will be called addressability, implying that members of the node can address members in a given domain unambiguously.

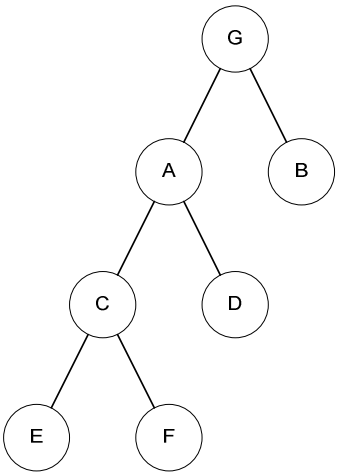

Figure 9. Domain tree.

This view of addressability is taken to represent the fact that members of a given sub-domain may be given addresses which are also used in another sub-domain. For example, domain G.A.C.E. may provide members with IP addresses in the standard local domain form 192.168.x.x. Domain G.A.C.F. may also use the local domain addressing mechanism. Thus, a member of G.A.C.E. may have an IP address 192.168.1.1, and so might a member of G.A.C.F. The domain tree structure ensures that this is not a problem.

As a sub-domain may address its parent domain, then it becomes obvious that a member of a parent domain may be connected to a member of a sub-domain. However, this connection must be initiated by the member of the sub-domain; connectivity is allowed down the domain tree but not addressability. For the purposes of discussion, messages can travel up or down the tree but not both in a single operation. A message travelling up or down must be received by a domain member before being sent in the other direction. This is normally handled by routers within normal network architectures but, as mobile channels are logical connections, an equivalent logical router is needed to redirect the message.

The analysis presented represents input channel end mobility, as this is the most complicated to achieve. For an input channel to be migrated, the architecture of the described model usually requires reconfiguration to ensure that messages are still received at the new input end location. For an output end, the majority of models permit the address or some other representation of the input end to be sent and a new output end to be created, effectively copying the output end at a new location. This is due to the Any-to-One architecture of a networked channel, where multiple output ends can connect to a single input end. Adding a new output end is trivial, and output end mobility involves adding a new output end and destroying the old one.

To aid in analysis, a number of values are defined. These are standard message types used in the underlying protocol to support CPA networking [15]:

- *proto* – a protocol message without any data. Acknowledgement messages are also considered protocol messages. As these messages should be of fixed size, communication time is constant.

- *addr* – the size of a channel location structure. These structures are used to permit the output end of a channel to connect to an input end. *addr* may vary based on implementation, thought communication time is considered constant.

- *msg* – a data message sent from one domain member to another. The size of *msg* is variable, and therefore communication time depends on message size.

To represent mobility, M_n is used, where n is the number of movement operations that have occurred since initial channel creation – M_0 representing a channel end that has not migrated.

Four properties of these models are investigated. These are *transmission time*, *reconfiguration time*, *reachability* and *robustness*.

3.1 Transmission Time

Transmission time is the time taken for a sent data message to arrive at its destination. This is an important Quality of Service (QoS) property in any distributed application, and is therefore an important value to analyse. The time taken to transfer a message of a particular type is expressed by the function t and is based on the amount of data sent. For the purposes of discussion a single communication between two domain members (even members in different domains in a branch), t is not affected by the actual distance up or down the domain tree travelled. A summary of these values is presented in Table 3. For simplicity, we assume that the transmission time for a message is independent of other messages being sent.

In all cases, a data message requires a subsequent acknowledgement, hence the *msg* and *proto* definitions within these equations.

Table 3. Transmission time.

Model	**Transmission Time**	**Description**
One-to-One	$M_n = t_{msg} + t_{proto}$	Connections are always direct.
Name Server	$M_n = t_{msg} + t_{proto}$ $[+ t_{msg} + t_{proto}]$	Connections are normally direct, although a connection may move thus requiring a resend.
Message Box	$M_0 = t_{msg} + t_{proto}$ $M_n = 2 \cdot t_{msg} + t_{addr} + t_{proto}$	First transmission is always direct. Subsequent messages require sending to message box and request from message box.
Message Box Server	$M_n = 2 \cdot t_{msg} + t_{addr} + t_{proto}$	As message box, although all sends are through the server.
Chain	$M_0 = t_{msg} + t_{proto}$ $M_n = n \cdot t_{msg} + n \cdot t_{proto}$	All messages travel the length of the chain.
Reconfiguring Chain	$M_0 = t_{msg} + t_{proto}$ $t_{msg} + t_{proto} \leq M_n \leq n \cdot t_{msg} + n \cdot t_{proto}$	With no reconfiguration messages travel the entire length of the chain. If reconfigured, there is the possibility of direct connections.
Mobile IP	$M_n = (up + down) \cdot t_{msg} +$ $(up + down) \cdot t_{proto}$	Messages travel through the domain agents up and down the domain tree.

3.2 Reconfiguration Time

Reconfiguration time is the time taken to reconfigure the communication architecture to permit the new communication path created by the migration of a channel. The time to reconfigure the architecture is another important QoS consideration and will affect

transmission time.

Reconfiguration complexity is represented by the parameter r that takes three values: *easy* for an architecture requiring little reconfiguration; *mod* for an architecture that requires some extra functionality and link creation; and *hard* for an architecture that requires a great deal of extra functionality and link creation to permit channel mobility. The time represented by r will generally be small in comparison to the time taken to transfer messages between nodes to allow reconfiguration.

Transfer time is taken into consideration for message transfer and acknowledgement. Channel transfer time for all models is either a protocol message or an address message, except for the reconfiguring chain which takes all previous addresses with it. Table 4 summarises.

Table 4. Reconfiguration time.

Model	**Reconfiguration Time**	**Description**
One-to-One	$M_n = r_{easy} + 2 \cdot t_{addr} + 2 \cdot t_{proto}$ $[+ t_{msg}]$	The sent mobile channel structure consists of an *addr* and acknowledgement, and this must also be sent to the companion channel end. A waiting message may also be sent with the channel.
Name Server	$M_n = r_{easy} + 6 \cdot t_{proto} + 2 \cdot t_{addr}$	The input end must send the new address to the server (ack'ed) and the client requests and receives this address (ack'ed).
Message Box	$M_n = r_{easy} + t_{addr} + t_{proto}$	Reconfiguration is simply sending the address to the new location with an acknowledged message.
Message Box Server	$M_n = r_{easy} + t_{addr} + t_{proto}$	As message box.
Chain	$M_n = r_{easy} + 2 \cdot t_{addr}$	The channel send contains the address and is acknowledged with the new address.
Reconfiguring Chain	$r_{easy} + 2 \cdot t_{addr} \leq M_n \leq r_{hard} + (n - 1) \cdot t_{addr}$	Worst case the channel end contains all previous addresses and must contact each to try and reconfigure. Best case is as chain.
Mobile IP	$M_n = r_{mod} + 2 \cdot (up + down) \cdot 2 \cdot t_{addr} + (up + down) \cdot t_{proto}$	The channel send contains two addresses (channel address and old address) and requires the new location to be sent back which contains two addresses. The send then must be acknowledged.

3.3 Reachability

Reachability is the set of domains where a channel input end can be hosted and a channel output end still successfully communicate to the input end within the defined model. This value is of interest as in theory we wish to send a channel end anywhere within a network and still provide connectivity between the input and output end. The problem lies in the domain architecture presented in Figure 9. For an output end to successfully connect to the input end, addressability must be possible. As addressability is only possible up a branch of the domain tree, supporting architecture is normally required to support full connectivity across the entire domain tree. To discuss reachability, three sets of domains are defined:

- *SUB_TREE* – the domain in which the input end of the channel is located, and all the sub-domains of this domain
- *BRANCH* – the set of domains within the same branch as the input end, implying both up and down traversal of the domain tree
- *GLOBAL* – the set of all domains

As it is possible for a node within a domain to connect up the tree, any model that allows such a connection is deemed to permit an output channel end that has migrated using an existing connection to be connected to an input channel end down the tree via the existing connection, although not the One-to-One model as shall be highlighted. Table 5 summarises reachability for the given models.

Table 5. Reachability.

Model	**Reachability**	**Description**
One-to-One	BRANCH (first) SUB_TREE ∩ BRANCH (subsequent)	The first interaction is always between connected domain members. Further migrations can only occur within the domain of the input end.
Name Server	SUB_TREE ∩ BRANCH	Although the server is SUB_TREE reachable, the input end must be reachable from the output end also. If the input end migrates to a separate branch, the output end cannot reach the input end, so reach is restricted to a branch of the domain.
Message Box	SUB_TREE □ BRANCH	The host of the message box can be told to connect up the tree to the new input location, and thus reachability is the union of SUB_TREE and BRANCH.
Message Box Server	SUB_TREE	As the server is a dedicated it would normally be connected to by the output and input end, and due to addressability gives reachability of SUB_TREE.
Chain	GLOBAL	The chain can stretch anywhere providing GLOBAL reachability.
Reconfiguring Chain	GLOBAL	As chain.
Mobile IP	GLOBAL	The chain of agents can stretch through all domains, providing GLOBAL reachability.

3.4 Robustness

The robustness of the model defines how strong the individual connections are between the input end and the output end of a channel. Robustness is defined by the reliance on external elements. A server (e.g. the message box model) is a stronger element than a normal networked node in the application due to the server being dedicated to supporting the network. Robustness is another key QoS property as if the connections between input and output ends are unreliable there is more chance that an application will fail.

For robustness there are three values to consider:

- *conn* – a connection between two nodes
- *node* – a normal node in the network
- *server* – a server node in the network

The robustness of the model depends on how many of these individual elements are required to support channel mobility. The fewer elements required by the model, the stronger the model. Table 6 summarises the robustness of the various models.

Table 6. Robustness.

Model	**Robustness**	**Description**
One-to-One	M_n = conn	There is only the connection between the input and output end.
Name Server	M_n = conn	The connection between the sender and receiver is always direct. A name server is required, but does not affect the robustness of the individual connection.
Message Box	M_0 = conn M_n = node + 2·conn	Initially, the model permits direct connection from sender to receiver. Then, connections from the individual ends to the message box are required.
Message Box Server	M_n = server + 2·conn	As message box, although a server is considered more robust than a hosting node. There is also no initial direct connection.
Chain	M_0 = conn M_n = n·(node + conn)	Initially the chain is directly connected. All subsequent migrations require the previous nodes to stay operational to provide connectivity.
Reconfiguring Chain	M_0 = conn conn ≤ M_n ≤ n·(node + conn)	The reconfiguring chain may be as weak as the normal chain in many regards, although a reconfiguration may result in a direct connection.
Mobile IP	M_0 = conn M_n = (up+down)·(conn+server)	Although a chain of agents is required for connectivity, these are considered dedicated entities in the architecture and thus provide moderate robustness to the connection backbone. Initially, the model provides a direct connection.

3.5 Summary

Table 7 summarises the different mobile channel models by placing them in order from best to worst under the respective property headings.

Taking these attributes together we can come to some firm categorisations of each of the models. These are not specific to certain hardware configurations, but are about the attributes that an application scenario might have as key considerations.

- *Best (if Any-to-One is not required)* – the One-to-One networked channel model provides the best transmission and reconfiguration times, as well as the strongest connectivity model. This comes at the cost of having restricted channel architectures, and this can especially be problematic for applications requiring a server type solution where multiple clients connect to a single server. Reachability is also poor.
- *Cluster* – the name server model provides good transmission time, reconfiguration time and robustness. Reachability is poor, but a cluster is in a centralised domain. If no Any-to-One connections are required, then the One-to-One model provides a better solution.
- *Global connectivity* – only three models provide global migration of channels but still allow connectivity between input and output channel ends. Of these, the two chain models are not strong and have high transmission times. Therefore, if global connectivity is required, the Mobile IP model is best. This comes at a cost of having a backbone of agents to handle routing and reconfiguration.

Table 7. Summary of mobile channel models

Transmission Time	**Reconfiguration Time**	**Reachability**	**Robustness**
One-to-One	One-to-One	Chain	One-to-One
Name server	Message box server	Reconfiguring chain	Message box server
Message box	Message box	Mobile IP	Name server
Message box server	Chain	Message box	Mobile IP
Reconfiguring chain	Name server	Message box server	Message box
Mobile IP	Mobile IP	One-to-One	Reconfiguring chain
Chain	Reconfiguring chain	Name server	Chain

So it can be seen that there is no one model which ideally suits all scenarios. As part of this work is to implement messages within the underlying network protocol to allow channel mobility between diverse frameworks, this becomes a problem as different frameworks generally have different application scenarios in mind. Further investigation into the protocol messages required is provided in the following section.

4. Protocol Integration

In this section only a brief discussion is presented, and a full discussion into the individual states and protocol messages required to support the various models is provided elsewhere [15].

In general, there are two important operations that must be supported by the protocol. The first is the migration of an input channel end (MIGRATE_INPUT), and the second is the migration of an output channel end (MIGRATE_OUTPUT). This provides the two most fundamental message types for each model. Subsequent to these two messages, there is a requirement for informing another entity of the arrival at a new location of a channel end, usually in the form of the address of the new channel location. This is based on the type of model being used. Beyond these most primitive message types, each model requires its own set of messages to support the reconfiguration of the underlying network architecture to support the migration of a channel end. Table 8 summarises the required protocol messages.

4.1 Summary

An analysis of the required protocol messages shows a number of commonalities between the separate models, which are in fact the three basic messages defined in at the beginning of this section. These three messages (*MIGRATE_INPUT*, *MIGRATE_OUTPUT* and *MOVED*) are common in the majority of models, and are the only messages in four of the models. Although adding these messages to the underlying network protocol may allow the usage of the separate models transparently, further work is required to discover if these models can all be supported within the protocol. Some of the messages require extra information within them to support the level of functionality, and the different states and underlying architectures may cause a problem. Therefore further examination within these areas is required.

5. Future Work

More analysis work in this area is required. The goal of this work is to provide an initial examination of these models in regards to suitability of supporting channel end mobility. From these attributes, scenarios can be developed that can be examined further within the context of the models presented.

Initially, simulation of each of the individual models in a suitable simulation environment is required. There are a number of network simulation tools available, and implementing each of these models within a simulator can help determine if any further messages or channel states are required to support the mobile channel architecture. The usage of a network simulator to generally simulate the underlying network protocol and architecture would also be advantageous, although some verification work has already been undertaken on JCSP Networking [15].

The individual models have yet to be implemented to examine the practical usage of each in real situations. This is one piece of work that must be carried out to determine whether or not the models are individually capable and suitable of supporting channel end mobility in a manner that is transparent to the user. The implementations can then have actual QoS properties measured and compared against the anticipated values. Actual required states and protocol messages can also be determined. Furthermore, suitable application models can be tested for suitability within each model.

Table 8. Protocol messages.

Protocol Message	Description	Models
MIGRATE_INPUT	Sent from the current hosting node of an input channel end to the new host node when an input channel end is migrated. Essentially a SEND of the input channel end.	All
MIGRATE_OUTPUT	As MIGRATE_INPUT but for an output channel end.	All
MOVED (a)	Sent to the companion channel end to inform of a location change.	One-to-One
MOVED (b)	Sent from the previous location of an input end to inform that the output end should resolve the new location of the input end from the name server.	Name Server
MOVED (c)	Sent from a node to a previous link to inform of the new location of the input channel end and that messages should be forwarded to this location. For the Mobile IP model this message is sent between the routing agents to reconfigure the channel path.	Chain Reconfiguring Chain Mobile IP
MOVING	Sent by the host of the input end to inform the name server that the channel end is about to move and that subsequent address resolutions should be buffered.	Name Server
ARRIVED	Sent by the receiver of an input end to inform the server that the input end has a new address and any pending resolutions may complete.	Name Server
RESOLVE	Sent to the server to request the current address of a given channel.	Name Server
RESOLVE_REPLY	Sent from the server as a response to the address resolution message.	Name Server
CHECK	Sent by the input end to check if any messages are waiting in the message box. This is required for guarded operations on the input channel.	Message Box Message Box Server
CHECK_RESPONSE	Sent in reply to a *CHECK* request. The response is immediate, although a later response may occur when a message appears in the message box. The message is dropped if the guarded operation completed prior to this..	Message Box Message Box Server
REQUEST	Request the next available message in the message box.	Message Box Server

Verification work is also required beyond the general simulation of the mobile channel models. Examining the models to ensure that they emit the behaviour that is expected, as well as examining the models for fundamental problems such as deadlock and livelock is important. More verification work on the new network protocol and architecture is also required to analyse behaviour.

Finally, work is still ongoing in regards to implementing the common protocol and a supporting architecture within the various CPA based distributed application frameworks. Work is ongoing with PyCSP[23] and the protocol is set to be implemented in Communicating Haskell Processes[24] and occam-π in the future. Work on a reduced JCSP version for small devices is also underway.

6. Conclusions

In this paper, we have presented a number of models that have the possibility of supporting distributed channel mobility in CPA based frameworks. Each of these models show promise in supporting the required functionality, but when analysed against some critical attributes such as message transmission time and robustness, it has been discovered that not all are fit for application scenarios that may require reliability or a certain level of Quality of Service. However, a number of models do show potential for supporting particular application scenarios very well, in particular the name server approach for cluster computing work. This does highlight that pony [7] did use the correct model considering its application area. The problem lies in finding a model that supports as many application scenarios as possible, which may be difficult.

To further support channel mobility, it is likely that a set of models is required, supported transparently by the underlying protocol. An analysis of the required protocol messages has highlighted three messages that are generally required, and a number of models that can be supported by a small number of protocol messages. Potentially, this means that the protocol can have channel mobility built in, and the underlying application architecture can support mobility in the manner best fitting the application scenario. Further work is required to analyse this potential further.

References

[1] K. Chalmers, J. Kerridge, and I. Romdhani, "A Critique of JCSP Networking," in *Communicating Process Architectures 2008*, P. H. Welch et al., Eds. Amsterdam, The Netherlands: IOS Press, 2008, pp. 271-291.

[2] G. P. Picco, "Mobile Agents: An Introduction," *Microprocessors and Microsystems*, 25(2), pp. 65-74, 2001.

[3] F. A. C. Polack, P. S. Andrews, and A. T. Sampson, "The Engineering of Concurrent Simulations of Complex Systems," in *2009 IEEE Congress on Evolutionary Computation*.: IEEE Press, 2009, pp. 217-224.

[4] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," *IEEE Personal Communications*, 8(4), pp. 10-17, 2001.

[5] P. H. Welch and F. R. M. Barnes, "Communicating Mobile Processes - Introducing occam-pi," in *Communicating Sequential Processes: The First 25 Years - Symposium on the Occasion of 25 Years of CSP*, A. E. Abdallah, C. B. Jones, and Sanders J. W., Eds. Berlin / Heidelberg, Germany: Springer, 2005, pp. 175-210.

[6] K. Chalmers, J. Kerridge, and I. Romdhani, "Mobility in JCSP: New Mobile Channel and Mobile Process Models," in *Communicating Process Architectures 2007*, A. A. McEwan et al., Eds. Amsterdam, The Netherlands: IOS Press, 2007, pp. 163-182.

[7] M. Schweigler and A. T. Sampson, "pony - The occam-pi Network Environment," in *Communicating*

Process Architectures 2006, P. H. Welch, J. Kerridge, and F. R. M. Barnes, Eds. Amsterdam, The Netherlands: IOS Press, 2006, pp. 77-108.

[8] A. Fuggetta, G. P. Picco, and G. Vigna, "Understanding Code Mobility," *IEEE Transactions on Software Engineering*, 24(5), pp. 342-361, 1998.

[9] J. Howell, "Straightforward Java Persistence Through Checkpointing," in *Proceedings of the 3rd International Workshop on Persistence and Java (PJW3): Advances in Persistent Object Systems*, D. Kotz and F. Mattern, Eds.: Morgan Kaufmann Publishers, Inc., 1999, pp. 322-334.

[10] D. Weyns, E. Truyen, and P. Verbaeten, "Serialization of Distributed Execution-state in Java," in *Objects, Components, Architectures, Services, and Applications for a Networked World: International Conference NetObjectDays, NODe 2002*, M. Aksit, M. Mezini, and R. Unland, Eds. Berlin / Heidelberg, Germany: Springer, 2003, pp. 41-61.

[11] W. Zhu, C.-L. Wang, W. Fang, and F. C. M. Lau, "A New Transparent Java Thread Migration System Using Just-In-Time Recompilation," in *The 16th IASTED International Conference on Parallel and Distributed Systems: PDCS 2004*, T. Gonzalez, Ed.: ACTA Press, 2004, pp. 766-771.

[12] R. Milner, J. Parrow, and D. Walker, "A Calculus of Mobile Processes, I," *Information and Computation*, 100(1), pp. 1-40, 1992.

[13] G.-C. Roman, G. P. Picco, and A. L. Murphy, "Software Engineering for Mobility: A Roadmap," in *Proceedings of the Conference on the Future of Software Engineering*.: ACM Press, 2000, pp. 241-258.

[14] A. Phillips, N. Yoshida, and S. Eidenbach, "A Distributed Abstract Machine for Boxed Ambient Calculi," in *Programming Languages and Systems: 13th European Symposium on Programming, ESOP*, D. Schmidt, Ed. Berlin / Heidelberg, Germany: Springer, 2004, pp. 155-170.

[15] K. Chalmers, "Investigating Communicating Sequential Processes for Java to Support Ubiquitous Computing," Edinburgh Napier University, Edinburgh, PhD Thesis 2009.

[16] H. Muller and D. May, "A Simple Protocol to Communicate Channels over Channels," in *Proceedings 4th International Euro-Par Conference: Euro-Par'98 Parallel Processing*, D. Pritchard and J. Reeve, Eds. Berlin / Heidelberg, Germany: Springer, 1998, pp. 591-600.

[17] M. Schweigler, "A Unified Model for Inter- and Intra-Process Concurrency," The University of Kent, Canterbury, PhD Thesis 2006.

[18] X. Zhong and C.-Z. Xu, "A Reliable Connection Migration Mechanism for Synchronous Transient Communication in Mobile Codes," in *International Conference on Parallel Processing 2004*.: IEEE Computer Society, 2004, pp. 431-438.

[19] A. R. Silva, D. D. Ramao, and M. M. da Silva, "Towards a Reference Model for Surveying Mobile Agent Systems," *Autonomous Agents and Multi-Agent Systems*, 4(3), pp. 187-231, 2001.

[20] J. M. Molina, J. M. Corchado, and J. Bajo, "Ubiquitous Computing for Mobile Agents," in *Issues in Multi-Agent Systems*, A. Moreno and J. Pavon, Eds.: Birkhauser Basel, 2007, pp. 33-57.

[21] F. Baude, D. Caromel, F. Huet, and J. Vayssiere, "Communicating Active Mobile Objects in Java," in *High Performance Computing and Networking: 8th International Conference, HPCN Europe 2000*, M. Bubak et al., Eds. Berlin / Heidelberg, The Netherlands: IOS Press, 2000, pp. 633-643.

[22] C. E. Perkins, "Mobile IP," *IEEE Communications Magzine*, 40(5), pp. 66-82, 2002.

[23] J. M. Bjorndalen, B. Vinter, and O. Anshus, "PyCSP - Communicating Sequential Processes for Python," in *Communicating Process Architectures 2007*, A. A. McEwan et al., Eds. Amsterdam, The Netherlands: IOS Press, 2007, pp. 229-248.

[24] N. C. C. Brown, "Communicating Haskell Processes: Composable Explicit Concurrency using Monads," in *Communicating Process Architectures 2008*, P. H. Welch et al., Eds. Amsterdam, The Netherlands: IOS Press, 2008, pp. 67-83.

[25] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed.: Addison Wesley, 2003.