A model for dynamically structured communicating systems

R.P. Hopkins and M. Koutny

Abstract

We here develop a model of structure and communication within systems of concurrently executing components connected by asynchronous communication links, and formulate a definition of observational equivalence between systems. The most important aspect of the model is that it accommodates the dynamic system connectivity required to model open and continuously operating systems, by allowing creation of new communication links and allowing an end-point of a communication link to be communicated as part of a message, with the recipient thus dynamically acquiring the ability to communicate on that link. For this model we formulate a novel definition of observational equivalence in terms of equivalent operational behaviours of possible observing systems. This formulation avoids certain difficulties that arise in attempting to apply the usual approach to dynamic systems, and also very directly captures the intuitive meaning of "observational equivalence".

Series Editor: M.J. Elphick

©1987 University of Newcastle upon Tyne.
Printed and published by the University of Newcastle upon Tyne,
Computing Laboratory, Claremont Tower, Claremont Road,
Newcastle upon Tyne, NE1 7RU, England.
Bibliographical details

HOPKINS, Richard Pinder

A model for dynamically structured communicating systems
[By] R.P. Hopkins and M. Koutny


(University of Newcastle upon Tyne, Computing Laboratory, Technical Report Series, no. 239.)

Added entries

KOUTNY, Maciej.
UNIVERSITY OF NEWCASTLE UPON TYNE.
Computing Laboratory. Technical Report Series. 239.

Abstract
We here develop a model of structure and communication within systems of concurrently executing components connected by asynchronous communication links, and formulate a definition of observational equivalence between systems. The most important aspect of the model is that it accommodates the dynamic system connectivity required to model open and continuously operating systems, by allowing creation of new communication links and allowing an end-point of a communication link to be communicated as part of a message, with the recipient thus dynamically acquiring the ability to communicate on that link. For this model we formulate a novel definition of observational equivalence in terms of equivalent operational behaviours of possible observing systems. This formulation avoids certain difficulties that arise in attempting to apply the usual approach to dynamic systems, and also very directly captures the intuitive meaning of "observational equivalence".

About the author

Dr. R.P. Hopkins has been a lecturer in the Computing Laboratory since October, 1983, prior to which he was a Research Associate.

Dr. M. Koutny has been at the Computing Laboratory as a Research Associate from March, 1985 until Nov. 1986 and as a Lecturer from December, 1986. Prior to this he was at Warsaw Technical University in Poland from 1982.

Suggested keywords

COMMUNICATION
CONCURRENCY
DYNAMICALLY STRUCTURED SYSTEMS
MESSAGE PASSING
OBSERVATIONAL EQUIVALENCE

Suggested classmarks (primary classmark underlined)
Dewey (18th): 001.64404
U.D.C. 519.687
A Model for Dynamically Structured Communicating Systems

by

Richard P. Hopkins and Maciej Koutny

Computing Laboratory, The University of Newcastle upon Tyne NE 1 7RU, U.K.

We here develop a model of structure and communication within systems of concurrently executing components connected by asynchronous communication links, and formulate a definition of observational equivalence between systems. The most important aspect of the model is that it accommodates the dynamic system connectivity required to model open and continuously operating systems, by allowing creation of new communication links and allowing an end-point of a communication link to be communicated as part of a message, with the recipient thus dynamically acquiring the ability to communicate on that link. For this model we formulate a novel definition of observational equivalence in terms of equivalent operational behaviours of possible observing systems. This formulation avoids certain difficulties that arise in attempting to apply the usual approach to dynamic systems, and also very directly captures the intuitive meaning of "observational equivalence".

1 INTRODUCTION

In this report we present a model of structure and communication within loosely-coupled systems, referred to as the DSCS (Dynamically Structured Communicating Systems) model. The major motivation for this model is to accommodate dynamically structured systems, by which we mean systems in which new components ("agents") can be created and the internal and external connectivity of a system can change during system operation. Such dynamic structure is we believe necessary to effectively model open and continuously operating systems which are subject to growth and change. For example a computer network application in which a mail server system must be able to acquire connections with similar systems which did not exist when it was first invoked, and must be able to pass on its current connections to a replacement system when the time comes to, say, upgrade its functionality.

The report is organised as follows. In the next section we introduce the model through an example showing its principle features. We then formalise the basic elements of the model, giving the operational semantics of systems. Finally we discuss the observable aspects of a system's behaviour, defining an observational equivalence and, in the appendix, give an example proof of equivalence of two systems. In our approach to observational equivalence two subsystems are equivalent if replacing one by the other in any larger system does not affect the operational semantics of the remainder of that larger system (that remainder being an "observer" which thus cannot detect the difference). In contrast, the usual approach (e.g. [MIL]) proceeds by first
defining an abstraction of a system's operational semantics which is asserted to give its observational semantics, and then defining equivalence of two systems as equivalence of their observational semantics. The novel approach adopted here has two motivations: firstly it deals very simply and effectively with certain difficulties that arise in attempting to apply the usual approach to systems with dynamically changing external connectivity; and secondly it gives a definition which seems to us to more directly capture the intuitive meaning of "observational equivalence".

Before proceeding with the development of the DSCS model we briefly discuss a number of particular characteristics in which various models of communication differ, in order to compare our model with others and motivate its particular characteristics. The other models covered are: "Communicating Sequential Processes" (CSP - [HOAb]); "Calculus of Communicating Systems" (CCS - [MIL]); "Actor Systems" (ACTORs - [HEW]); and "Networks of Stream Functions" (STREAMs - [KAH]).

**Port-based vs. agent-based communication.** In all models a system consists of active agents participating in events such as sending messages to other agents. In CCS, STREAMs and CSP of [HOAb] there are ports which define the communication channels between agents; the destination/source of a message is identified in terms of a particular port. In contrast, in ACTORs and CSP of [HOAa], the source/destination of a message to be received/sent is identified in terms of the other agent involved in the communication. In the case of the DSCS model we have adopted port-based communication. Port-based communication facilitates good software engineering by allowing the decoupling of particular interfaces (each provided at a particular port) from the actual agent which happens to implement one or more of these interfaces (by responding to messages communicated on the appropriate port). Also, with port-based communication there is a general ability to construct agents as compositions of other agents, whereas this ability is compromised in agent-based communication due to the impossibility of there being multiple distinct communication channels between the same pair of agents. For example, if there are four agents $A_1 - A_4$ with a connection from $A_1$ to $A_2$, and an independent connection from $A_3$ to $A_4$, then in composing $A_1$ and $A_3$ as one agent, and $A_2$ and $A_4$ as one agent, the independence of the connections would be lost in an agent-based model.

**Static vs. dynamic configuration.** The CCS, CSP and STREAMs model are all statically configured, that is they require that for the system being modelled there be a complete pre-existing program text which explicitly defines the full (potential) connectivity between all the (potential) agents in the system and fixes the communication connections with the system's external environment. In contrast, the DSCS model and the ACTORs model allow a system to dynamically create internal connections, acquire connections with its external environment and internally reconfigure existing connections; thus being able to directly model important classes of systems which cannot be effectively handled by statically configured models.
Deterministic vs. non-deterministic models. Most models, including DSCS, are non-deterministic in that for example there can be several input messages available for an agent such that one will be received first but which one is not predictable. This is a commonly occurring situation in many distributed computing systems and the inability of a deterministic model such as STREAMs to handle it is recognised as a major limitation.

Private vs. shared ports. In, for example, CCS, a port can be freely shared between a number of agents and thus it may be that several agents $A_i$ can send a message on the same communication channel and several agents $B_j$ can receive a message sent on that channel. A message sent by one of the $A_i$'s may be received by any one of the $B_j$'s, the sender having no control nor implicit knowledge of the message's destination agent, and the receiving agent having no control nor implicit knowledge of its source. Also there may be the need for non-local resolution of non-deterministic choices within the various agents sharing a channel. In contrast, in our model at most one agent can output to a communication channel and at most one agent can input from it (although which particular agent can do so may vary as a result of the dynamic configuration property).

Synchronous vs. asynchronous models. In CSP and CCS a communication event is the instantaneous transfer of a message, requiring synchronisation between sending and receiving agents. In contrast, in STREAMs and ACTORS the communication involves separate unsynchronised events for the sending and receiving of the message by the two agents. For the DSCS model we adopt asynchronous communication as this is more natural and effective for the fully decentralised systems which are our principal interest. By adopting asynchronous communication and private ports we ensure that any non-deterministic choice can only involve a single agent and thus can always be resolved locally at that agent. In contrast, non-deterministic models with synchronous communication (and/or shared ports) may require non-local resolution of non-deterministic choices, which is complex to implement in a fully decentralised system.

In summary, the main features of the DSCS model are: port-based asynchronous communication; dynamic systems structure; and non-deterministic behaviour. Compared with the ACTORS model the main difference is port-based rather than agent-based communication. Compared with the remaining models the main difference is in allowing dynamic system structure although the use of private rather than shared ports is also significant.

2 OBJECTS, PORTS AND EVENTS

Here we illustrate the basic features of the DSCS model by following through an example, shown in Figure 1. This will be interpreted in terms of a simple network filing system in which the creation of a file results in a new file agent $F$ (which holds the file data and services access requests), the log-in of a user results in a new user agent $U$, and the opening of a file by a user establishes a new connection between those agents. In addition there is a system manager agent,
$M$, which dynamically creates the user and file agents and can route a message from a user to an identified file.

![Diagram of configurations and transitions](image)

**Figure 1**

There is a succession of nine configurations $C_1$-$C_9$ with transitions between configurations being by events $E_1$-$E_8$. There are: agents, represented by circles; and messages, represented by boxes. Agents and messages (generically "objects") can own ports, each port being an *inport* or *outport*. An inport-outport pair of corresponding ports defines a link from the object owning the outport to the object owning the inport. For example, in configuration $C_2$ there is a directed arc, labelled $a$, which represents a link connecting $U$ to $M$, i.e. $U$ owns the $a_{outport}$ and $M$ owns the corresponding $a_{inport}$. 
In configuration $C_I$ there is a single "system manager" agent $M$. In event $E_4$, a "user log-in", $M$ creates a new link $a$ and a new "user process" agent $U$, giving the $a_{outport}$ to $U$ and retaining the $a_{inport}$. In the next event, "file creation", the manager creates a new "file" agent $F$, with a link $b$ from $M$ to $F$, as shown in $C_3$. In event $E_3$, the user, using its $a_{outport}$, sends to $M$ a message, $\mu$, which owns two ports: the $a_{outport}$, and the $c_{inport}$ which is part of a new link created in this event. The $a_{outport}$ is the message’s destination outport, identifying its destination - a message can only be received by an agent, in this case $M$, which owns the corresponding inport. The $c_{inport}$ conveyed by that message will allow further communication between $U$ and $M$, as seen in the next few events.

In event $E_4$, the user sends another message, $\mu'$, which is a request to "open" file $F$. The ports of this message are: $c_{outport}$ as its destination outport; the $d_{inport}$ serving the same role as the $c_{inport}$ of $\mu$; the outport of a new link $e$, and the import of a new link $f$ - these will eventually ($C_9$) provide direct bi-directional communication between the user and the file. In general a message can have several outports (e.g. $\mu'$ has $c_{outport}$ and $e_{outport}$) one of which ($c_{outport}$) is distinguished as the message’s destination outport. Event $E_4$ gives the situation of $C_5$ in which there is a sequence of two outstanding messages, $\mu$ and $\mu'$ from $U$ to $M$. When in event $E_5$ the manager receives $\mu$, it acquires the $c_{inport}$ and thus the ability to receive $\mu'$.

In receiving a message, $\mu$ or $\mu'$, the link which made the communication possible, i.e. $a$ or $c$, is (necessarily) destroyed, and the content of the message is acquired by the receiving agent. In the case of $E_6$, the received message is in $E_7$ re-transmitted to $F$, taking with it the ports of links $e$ and $f$. When the file receives the message $\mu'$, we have the situation $C_9$ in which the connectivity of $C_9$, where the user was only indirectly connected to the file, has been extended by two links, $e$ and $f$, allowing direct, bi-directional communication between those agents without further use of the intermediary agent $M$. A sequence of messages can now be sent from $U$ to $F$, in the same way as $\mu$ and $\mu'$ were sent from $U$ to $M$. As $F$ has now two inports, it is possible for there to be simultaneously two messages available for input, and $F$'s "program" may involve non-deterministic choice of which to input.

There are a number of points to be noted in the outline of the model presented above and the way it is formalised subsequently.

(i) We adopt absolutely the principle that all communication must be explicitly modelled - an agent's behaviour can be influenced only by the receipt of a message on one of its inports. This principle requires that the formalism be able to deal with an "open" system having external links, that is links for which one port is owned by some object in the system and the other port is not. A system without such external links (a closed system) cannot influence or be influenced by anything outside itself, thus being of no practical significance.

(ii) Every message has, if nothing else, a destination outport acquired from the sending agent, and can only be received by an agent owning the corresponding inport. A message or agent can of course have data as well as ports (for this example the data for $\mu$ would be some file identifier, and
the data for \( M \) would include a directory mapping file identifiers to outports).

\((iii)\) In any system configuration at most one agent or message can own a particular port. In the formalisation we consider there to be fixed set of ports that can be used for communication links. Bringing a port into use for a link, as occurs for \( c_{\text{outport}} \) and \( c_{\text{inport}} \) in \( E_3 \), will be referred to as activating it; taking a port out of use, as occurs for those ports in \( E_6 \), will be referred to as deactivating it. Two corresponding ports defining a link will always be activated/deactivated together, in the creation/destruction of that link.

\((iv)\) A single mechanism of two objects being linked by owning corresponding ports is used to model both the relationship between messages (their sequence) and the relationship between agents (their communication links).

\((v)\) Each event (e.g. \( E_5 \)) will have one or two predecessor objects, a principal agent \( (M) \) and possibly an input message \( (\mu) \); and a number (possibly zero) of successor objects, comprising zero or more agents and zero or more output messages. In, say, \( C_3 \), we informally consider \( M \) to be a new version of \( M \) in \( C_2 \) and \( F \) to be a completely new agent. Formally however both \( M \) and \( F \) of \( C_3 \) are new agents created in \( E_2 \), and \( M \) of \( C_2 \), \( E_2 \)'s principal agent, disappears in that event.

### 3 SYSTEM STRUCTURE AND EVOLUTION

In this section we formalise the basic elements of the DSCS model (ports, agents and messages), and introduce behaviour specifications prescribing events in which the agents can participate. We then define systems and their operational semantics, and introduce the notion of the composition of two systems.

#### 3.1 Ports and Generators

There is an infinite set of ports \( P \), partitioned into an infinite set of inports \( P_{IN} \) and an infinite set of outports \( P_{OUT} \). For each inport (outport) \( p \) there is a corresponding outport (inport) which will be denoted by \(-p\) or, simply, by \(-p\).

**Definition 3.1.1** The operator, \(-: P \rightarrow P\), is a mapping satisfying:

\[-(P_{IN}) = (P_{OUT}); -(P_{OUT}) = (P_{IN}); -(p) \neq -(q); \text{ and } -(\{p\}) = p, \text{ for all } p \neq q \in P.\]

As a system evolves an agent can activate an inport-outport pair of corresponding ports as a new communication link. In order to ensure that every such new communication link is unique, we associate with every agent a unique (port) generator \( g \) modelled as an infinite sequence of distinct outports, \((g,i)\) denoting the \( i \)-th outport in the sequence. In an event involving a principal agent with generator \( g \), some number \( k (k \geq 0) \) of new links will be created, for which are used the initial \( k \) outports of \( g \), \((g,1), \ldots, (g,k)\), together with their corresponding inports, i.e. \(-g(1)\), \ldots, \(-g(k)\). The remaining ports are split into disjoint infinite subsequences which provide the (unique) generators for the successor agents resulting from that event.
Definition 3.1.2 A generator, \( g \in \text{Gen} \), is an infinite sequence of outports \( g = (g_i, g_x, \ldots) \) for which \( g_i \neq g_j \) for all \( i \neq j \).

For every \( i \geq 1 \) we let \((.,i) : \text{Gen} \rightarrow P\text{OUT} \) be a mapping such that:

\[
(g,i) = g_i \text{ for every } g = (g_1, g_2, \ldots) \in \text{Gen}.
\]

To express the formation of the generators for the successor agents we introduce the notation \([j:k](g)\) to denote the \( j \)-th generator which may be derived from the generator \( g \) after deleting the first \( k \) outports. To illustrate the use of this notation, if \( M \) in configuration \( C_1 \) of Figure 1 has a generator \( g \), then in \( C_2 \): the ports forming link \( a \) will be \((g,1)\) owned by agent \( U \) and \(-(g,1)\) owned by agent \( M \); the generator for agent \( U \) will be \([1:1](g)\); and that for the new version of \( M \) will be \([2:1](g)\).

Definition 3.1.3 There is a family of mappings \([j:k] : \text{Gen} \rightarrow \text{Gen} \) for every \( j \geq 1 \) and every \( k \geq 0 \), such that if \( j,i \geq 1, j \neq i, k \geq 0 \) and \( g \in \text{Gen} \), then:

(i) \([j:k](g)\) is a sequence of elements of \( g \) excluding \((g,1), \ldots, (g,k)\);

(ii) \([j:k](g)\) and \([i:k](g)\) have no elements in common.

Note that a possible function for \([j:k](g)\) is:

\[
[j:0](g) = ((g,2i-1), \ldots, (g,(2i+1)2i-1), \ldots); \quad \text{and}
\]

\[
[j:k](g) = [j:0](g,k+1),(g,k+2), \ldots).
\]

3.2 Agent and Message Expressions

A system will consist of a (finite) set of (initial) objects (agents and messages) and a (possibly infinite) set of behaviour specifications which determine the possible transitions of the system to a new system with a modified set of objects (but the same set of behaviour specifications). An agent, \( a \), is a quintuple, \( a = (y; \{i; \text{out}; x\}) \), where the \text{in}, \text{out} and \( x \) are finite (possibly empty) sequences of input and output ports (by which the agent is linked to other objects), and data values (comprising the agent's state). The \( g \) is a port generator and the \( y \) is a behaviour identifier, drawn from an infinite fixed set ID, which as will be seen determines which of the system's behaviour specifications are applicable to that agent. A message, \( \mu \), is a triple \( \mu = (\text{in}; \text{out}; x) \), where \text{in}, \text{out} and \( x \) are as before, except that we refer to the \( x \) as the message's content; and \text{out} is non-empty with its first element, denoted by \( \text{dest}(\mu) \), being the message's destination outport which it will be recalled determines which agent can receive the message.

A behaviour specification defines a family of possible events using values, variables and expressions for objects and their constituent ports etc., such that a suitable substitution of values for variables in a specification gives a characterisation of one possible event for the system. We will use the notation \text{expr}/f to mean the result of applying a substitution \( f \) (i.e. a function assigning values to variables of appropriate types) to an expression \text{expr}. The various types of
elements employed in behaviour specifications are as follows, in which \( \text{Var}_{\text{bool}}, \text{Var}_{\text{int}}, \text{Var}_{\text{inp}}, \text{Var}_{\text{outp}} \) and \( \text{Var}_{\text{gen}} \) are disjoint, infinite sets of variables.

**Data** For the data items constituting the content of a message and state of an agent we employ data values, variables and expressions of types boolean \( \{ \text{TRUE}, \text{FALSE} \} \), \( \text{Var}_{\text{bool}} \) and expressions \( \text{Expr}_{\text{bool}} \) and integer \( \{ \ldots -1,0,1 \ldots \} \), \( \text{Var}_{\text{int}} \) and \( \text{Expr}_{\text{int}} \). In boolean and integer expressions we use the usual operators on boolean and integer values and variables.

**Port & Generator** For the ports owned by an agent or message and the generator associated with an agent, we employ import, output and generator values \( (P_{\text{IN}}, P_{\text{OUT}}, \text{Gen}) \), variables \( (\text{Var}_{\text{inp}}, \text{Var}_{\text{outp}}, \text{Var}_{\text{gen}}) \) and expressions \( (\text{Expr}_{\text{inp}}, \text{Expr}_{\text{outp}}, \text{Expr}_{\text{gen}}) \). For port and generator expressions we use the "-", ",", ".", and "[j:k]" function symbols introduced above.

**Agent & Message** For the agents and messages participating in an event, we employ agent and message expressions, where an agent expression \( \alpha \in \text{Expr}_\text{ag} \) is any quintuple \( \alpha \in (\text{ID} \times \text{Expr}_\text{gen} \times \text{Expr}_{\text{inp}} \times \text{Expr}_{\text{outp}} \times \text{Expr}_\text{int}) \), and a message expression \( \mu \in \text{Expr}_\text{mess} \) is any triple \( \mu \in (\text{Expr}_{\text{inp}} \times \text{Expr}_{\text{outp}} \times \text{Expr}_\text{int}) \). The various elements of an agent quintuple, represented \( \alpha = (y; g; \text{in}; \text{out}; \chi) \), and a message triple, \( \mu = (\text{in}; \text{out}; \chi) \), have the interpretation described above, with the first element of the output expression sequence of \( \mu \) being denoted by \( \text{dest}(\mu) \).

The above characterisation of agent and message expressions covers both behaviour specifications and agent and message instances which occur as actual objects of a system. The latter are expressions with no variables, and thus with every constituent expression giving a uniquely defined value. Thus, using \( \text{Var(expr)} \) to denote, for any expression \( \text{expr} \), the set of all variables appearing within it, we have:

**Definition 3.2.1** An agent is any agent expression \( \alpha \in \text{Expr}_\text{ag} \) such that \( \text{Var}(\alpha) = \emptyset \). A message is any message expression \( \mu \in \text{Expr}_\text{mess} \) such that \( \text{Var}(\mu) = \emptyset \). The sets of agents and messages will be denoted by \( \text{Ag} \) and \( \text{Mess} \), respectively.

### 3.3 Communication Specifications

There are two types of behaviour specifications, the first dealing with communication events, e.g. \( E_\delta \) of Figure 1, in which an agent receives an input message. The specification of a communication event will be represented as:

\[
a_0 \quad \mu_0 p \quad \text{guard} \rightarrow \quad a_1 \ldots a_m \quad \mu_1 \ldots \mu_n
\]

Here \( a_0 \) is an agent expression for the principle agent of a possible event in which an input message represented by message expression \( \mu_0 \) is received on a port \( p \) of \( a_0 \), to produce zero or more successor agents represented by agent expressions \( a_1, \ldots, a_m \), and zero or more output messages represented by message expressions \( \mu_1, \ldots, \mu_n \). The guard is a boolean expression restricting the applicability of the specification.
As an example we will consider the configuration $C_\delta$ of Figure 1 with an arbitrary choice of those details, such as data values in an agent’s state, which are not represented in the figure. The agent $M$ is the variable-free agent expression:

$$\alpha: \quad (ID_M; (g_{output}; g_{output} ; \ldots); c_{import}; b_{output}; 1)$$

where $ID_M$ is $M$’s behaviour identifier, $(g_{output}; g_{output} ; \ldots)$ is $M$’s generator, $c_{import}$ and $b_{output}$ are the ports by which $M$ is linked to other objects ($\mu'$ and $F$) and the $I$ is a singleton sequence for $M$’s state. The message $\mu'$ is the variable-free message expression:

$$\mu: \quad (d_{import}; f_{import}; c_{output}; e_{output}; 20)$$

where $20$ is the message content and the other elements are the ports by which the message is linked to other objects. The following is a possible behaviour specification, $A$, (using variables $h, p, q, x_1, r, s, t, u, x_2$) for the system to allow a direct transition from $C_\delta$ to $C_\delta$:

$$A: \quad (ID_M; [1:1](h); r; (h.1); x_1 + 1) ; (\langle h.1 \rangle, s; q, u; x_2)$$

The above specification (of form: $a_0 \cdot \mu_0; p \cdot guard \rightarrow a_1 \cdot \mu_1$) is applicable to a particular agent $M$ and particular message $\mu'$ if there is a substitution $f$ of values for variables in $a_0$ and $\mu_0$ such that $a_0$ evaluates to $M$, $\mu_0$ evaluates to $\mu'$, the guard evaluates to TRUE, and $p$ and dest($\mu_0$) evaluate to the input and output port of the same link (i.e. $p/f = -dest(\mu_0/f)$). Clearly there is such a substitution, $h/f = (g_{output}; g_{output} ; \ldots)$, $p/f = c_{import}$, $q/f = b_{output}$, $x_1/f = 1$, $r/f = d_{import}$, $s/f = f_{import}$, $t/f = c_{output}$, $u/f = e_{output}$ and $x_2/f = 20$, and so the specification can be applied. The effect of applying the specification is to replace $M$ and $\mu'$ by those agents and messages given by applying the same substitution $f$ to the agent and message expressions in the remainder of the specification, that is

$$\alpha_b; \mu: \quad (ID_M; [1:1](h); r; (h.1); x_1 + 1) \rightarrow (\langle h.1 \rangle, s; q, u; x_2)$$

giving

$$\alpha_b; \mu: \quad (ID_M; (g_{output}; g_{output} ; \ldots); d_{import}; f_{import}; b_{output}; 2) \rightarrow (g_{import}; f_{import}; b_{output}; e_{output}; 20)$$

which are variable-free expressions for agent $M$ and message $\mu'$ in $C_\delta$ (with other objects, $U$ and $F$, being carried forward unchanged). The event for this transition from $C_\delta$ to $C_\delta$ can be represented as a pair $(U, V)$, where $U$ is a set comprising the original agent and message instances, $U = \{a, \mu\}$; and $V$ is the set of resulting agent and message instances, $V = \{a_b, \mu_b\}$. The meaning $\Psi(A)$ of this behaviour specification, $A$, will be given as the (infinite) set of all possible events prescribed by it, i.e. all those pairs $(U, V)$ such that $A$ can be applied to $U$, producing $V$.

This example illustrates the use of behaviour identifiers to achieve the association between an agent instance in a system and the behaviour specifications applicable to it - the above behaviour specification is clearly only applicable to agents such as $M$ which have the same behaviour identifier $ID_M$ as in the $a_0$ of the specification. There may in fact be several different specification applicable to the same agent, e.g. in addition to the above specification we might
have the specification:

\[ (ID_M; h; p; q; x_1) \rightarrow (ID_M; \{1.0\}(h); r, s; q, u, x_1, x_2) \]

A possible event for this, using the same substitution function as above would be \( E_\delta \) of Figure 1. Considering these two specifications together we see that if the integer \( x_2 \) of the input message is exactly 0, then we have non-determinism in the behaviour of agent \( M \) since both of the above specifications are applicable and give different results.

We now formally define a communication specification \( A_c \) and its meaning \( \Psi(A_c) \), using the notations \([s]\) to denote the set of all elements appearing in any sequence \( s \), and \#(s) to denote the length of a finite sequence \( s \):

**Definition 3.3.1** A communication behaviour specification \( A_c \) is defined to be any sequence of expressions \( A_c \in \text{Expr}_{ag} \times \text{Expr}_{mess} \times \text{Expr}_{inp} \times \text{Expr}_{bool} \times \text{Expr}_{ag}^* \times \text{Expr}_{mess}^* \) such that if these expressions are denoted by \( a_0, \mu_0, p, \text{guard}, a_1 \ldots a_m, \mu_1 \ldots \mu_n \), where for \( 0 \leq i \leq m \) and \( 0 \leq j \leq n \),

\[
a_i = (y_i; g_i; i_{in_i}; i_{out_i}; x_i) \quad \text{and} \quad \mu_j = (i_{in_0,j}; i_{out_0,j}; x_{0,j}),
\]

then for some \( k \geq 0 \) (the number of new links created in the event):

**A.1** \( p \in [\text{in}_0] \) and, if \( \Omega = [\text{in}_0] \cup [\text{out}_0] \cup [x_0] \cup [i_{in_0,0}] \cup [i_{out_0,0}] \cup [x_{0,0}] \), then:

\[ \Omega = \text{Var}(a_0) \cup \text{Var}(\mu_0), \]

and \( |\Omega| = 1 + \#(i_{in_0}) + \#(i_{out_0}) + \#(x_0) + \#(i_{in_0,0}) + \#(i_{out_0,0}) + \#(x_{0,0}) \);

**A.2** \( \text{Var}({\text{expr}}) \subseteq \Omega \) for every \( \text{expr} \in \{\text{guard}, a_1 \ldots a_m, \mu_1 \ldots \mu_n\} \);

**A.3** if \( \Phi = ([\text{in}_1] \cup [\text{out}_1]) \ldots ([\text{in}_m] \cup [\text{out}_m]) \cup ([\text{in}_0,1] \cup [\text{out}_0,1]) \ldots ([\text{in}_0,n] \cup [\text{out}_0,n]) \) then:

\[ \Phi = [\text{in}_0] \cup [\text{out}_0] \cup [i_{in_0,0}] \cup [i_{out_0,0}] \cdot \{p, \text{dest}(\mu_0)\} \cup \{(g_0,1),\ldots,(g_0,k),-\,(g_0,1),\ldots,-(g_0,k)\}, \]

and \( |\Phi| = \#(i_{in_0}) + \#(i_{out_0}) + \#(i_{in_0,0}) + \#(i_{out_0,0}) \cdot 2 + 2k \);

**A.4** \( g_j = [j:k](g_0) \) for \( j = 1, \ldots, m \).

That is, we have: (A.1) \( p \) is an import variable of \( a_0 \) and except for \( a_0 \)'s behaviour identifier, \( a_0 \) and \( \mu_0 \) comprise just variables, no such variable appearing more than once; (A.2) all variables in \text{guard}, \( a_1 \ldots a_m \) and \( \mu_1 \ldots \mu_n \) are in \( a_0 \) and \( \mu_0 \); (A.3) the port expressions in \( a_1 \ldots a_m \) and \( \mu_1 \ldots \mu_n \) are exactly all port variables of \( a_0 \) and \( \mu_0 \) except \( p \) and \( \text{dest}(\mu_0) \), together with expressions: \( (g_0,1),\ldots,(g_0,k) \) and \( -(g_0,1),\ldots,-(g_0,k) \) for some \( k \geq 0 \); each such variable or expression appearing exactly once; (A.4) the generator expression of \( a_j \) is \( [j:k](g_0) \) for \( j = 1, \ldots, m \).

**Definition 3.3.2** Let \( A_c \) be a behaviour specification as in Definition 3.3.1. We define \( \Psi(A_c) \) to be the set of all pairs \( (U, V) = ([a_0], [a_1],\ldots,[a_m], [\mu_0], [\mu_1],\ldots,[\mu_n], f) \), where \( f \) is a substitution function with the domain \( \text{Var}(a_0) \cup \text{Var}(\mu_0) \) and such that:

\[ \text{guard}/f = \text{TRUE} \quad \text{and} \quad -p/f = \text{dest}(\mu_0)/f. \]

Clearly, condition A.2 on behaviour specifications implies that \( V \) is a set of agent and message instances (i.e. the substitution makes those expressions variable-free). Also, since the boolean and
integer expressions cannot use port or generator variables, the data values forming the states of the successor agents and contents of the output messages depend only on the data values forming the state of principle agent \( a_0/f \) and the content of input message \( \mu_0/f \), that is:

**Corollary 3.3.3** Let \( A_3 \) be a behaviour specification as in Definition 3.3.1, and let \( e,f \) be two substitution functions with the domain \((\text{Var}(a_0) \cup \text{Var}(\mu_0))\) such that:

\[
guard/e = \text{TRUE}; \quad -p/e = \text{dest}(\mu_0/e); \quad -p/f = \text{dest}(\mu_0/f); \quad x_0/e = x_0/f; \quad \text{and} \quad x_0,0/e = x_0,0/f.
\]

Then: \( \text{guard}/f = \text{TRUE} \); for every \( 1 \leq i \leq m \), \( x_i/e = x_i/f \); and for every \( 1 \leq j \leq n \), \( x_0,j/e = x_0,j/f \).

\( \Box \)

### 3.4 Spontaneous Behaviour Specifications

In addition to the communication behaviour specifications described above, we have a class of spontaneous behaviour specifications prescribing events, such as \( E_1 \) or \( E_2 \) in Figure 1, in which the principal agent sends new messages and/or creates successor agents without receiving a message. We give here just the formal definitions which are directly analogous to those in the previous section.

**Definition 3.4.1** A **spontaneous behaviour specification** \( A_s \) is defined to be any sequence of expressions \( A_s \in \text{Expr}^{\text{ag}} \times \text{Expr}^{\text{boot}} \times \text{Expr}^{\text{ag}}_* \times \text{Expr}^{\text{mess}}_* \) such that if these expressions are denoted by \( a_0, \text{guard}, a_1, ..., a_m, \mu_I, ..., \mu_n \), where for \( 0 \leq i \leq m \) and \( 1 \leq j \leq n \),

\[
a_i = (\gamma_i; g_i; in_i; out_i; x_i) \quad \text{and} \quad \mu_j = (\text{in}_0,j; out_0,j; x_0,j),
\]

then for some \( k \geq 0 \):

**A.5** if \( \Omega = \{g_0\} \cup \{\text{in}_0\} \cup \{\text{out}_0\} \cup \{x_0\} \), then:

\[
\Omega = \text{Var}(a_0),
\]

and \( |\Omega| = 1 + \#(\text{in}_0) + \#(\text{out}_0) + \#(x_0) \);

**A.6** \( \text{Var}^{\text{expr}}(\text{expr}) \subseteq \Omega \) for every \( \text{expr} \in \{ \text{guard}, a_1, ..., a_m, \mu_I, ..., \mu_n \} \);

**A.7** if \( \Phi = \{\text{in}_1\} \cup \{\text{out}_1\} \cup \ldots \cup \{\text{in}_m\} \cup \{\text{out}_m\} \cup \{\text{in}_0,1\} \cup \{\text{out}_0,1\} \cup \ldots \cup \{\text{in}_0,n\} \cup \{\text{out}_0,n\} \) then:

\[
\Phi = \{\text{in}_0\} \cup \{\text{out}_0\} \cup \{\langle g_0,1 \rangle, ..., \langle g_0,k \rangle, \langle g_0,1 \rangle, ..., \langle g_0,k \rangle \},
\]

and \( |\Phi| = \#(\text{in}_0) + \#(\text{out}_0) + 2k \);

**A.8** \( g_j = \{j,k\}(g_0) \) for \( j = 1, ..., m \).

**Definition 3.4.2** Let \( A_s \) be a behaviour specification as in Definition 3.4.1. We define \( \Psi(A_s) \) to be the set of all pairs \( (U,V) = (\{a_0/f\}, \{a_1/f, ..., a_m/f, \mu_I/f, ..., \mu_n/f\}) \), where \( f \) is a substitution function with the domain \( \text{Var}(a_0) \) and such that \( \text{guard}/f = \text{TRUE} \).

**Corollary 3.4.3** Let \( A_s \) be a behaviour specification as in Definition 3.4.1, and let \( e,f \) be two substitution functions with the domain \( \text{Var}(a_0) \) such that: \( \text{guard}/e = \text{TRUE} \) and \( x_0/e = x_0/f \).

Then, \( \text{guard}/f = \text{TRUE} \); for every \( 1 \leq i \leq m \), \( x_i/e = x_i/f \); and for every \( 1 \leq j \leq n \), \( x_0,j/e = x_0,j/f \).

\( \Box \)
3.5 The Construction of Systems

Having defined a single behaviour specification applicable to a single principle agent (and input message), we now introduce a system as the composition of a set of agents and messages, the system's initial objects, and a finite or countably infinite set of behaviour specifications, referred to as a script. There is a crucial restriction on valid systems that no port appear as part of more than one object, that is the property that there is no sharing of communication links. We now define the notion of an admissible set of objects for which this restriction is satisfied, first introducing the notations \( \alpha_p \) for the sequence of ports owned by an agent instance \( \alpha \) (including those in its generator), \( \mu_p \) for the sequence of ports owned by a message \( \mu \), and \( P(W) \) for the set of ports owned by any set of objects \( W \).

**Definition 3.5.1** With every agent \( \alpha = (\gamma;g;in;out;x) \in Ag \) we associate a sequence denoted by \( \alpha_p \) and defined as the concatenation of \( in \) and \( out \) and the sequence \((g.1),-(g.1),((g.2),-(g.2),...\).

With every message \( \mu = (in;out;x) \in Mess \) we associate a sequence denoted by \( \mu_p \) and defined as the concatenation of \( in \) and \( out \).

**Definition 3.5.2** A set \( W \subseteq (Ag \cup Mess) \) is said to be admissible iff for every \( w \in W \), \( wp \) is a sequence of distinct elements; and for all \( w \neq v \in W \), \( wp \) and \( vp \) do not have an element in common. We denote this by \( W \in Adm \).

For a set of objects \( W \) we use \( P(W) \) to denote the set of all ports appearing in the sequences \( wp \), for all \( w \in W \).

We now define the general notion of a configuration \( C \) as any set of objects \( W \), together with any script \( \Gamma \); and the restricted notion of a system as a configuration in which \( W \) is a finite admissible set of initial objects.

**Definition 3.5.3** A configuration is defined to be any pair \( C = <W;\Gamma> \), for which \( W \subseteq (Ag \cup Mess) \) and \( \Gamma \) is a script.

A configuration \( C = <W;\Gamma> \) is said to be a system iff \( W \in Adm \) and \( W \) is finite.

For a system \( C = <W;\Gamma> \) we use \( P(C) = P(W) \) to denote the set of ports owned by all objects of that system. Also, for a configuration or a system \( C \) we use \( WC \) and \( \Gamma_C \) to denote its first and second element, respectively.

A system is said to be an open system if there are any port pairs (i.e. communication links) for which one port, \( p \), is owned by (an object of) the system, but the other, \(-p\), is not; port \( p \) being referred to as an external port of the system. An external input/output is part of a communication link which goes outside the system and thus potentially allows a message to be input from (output to) the system's external environment. If port \( q \) and its corresponding port \(-q\) are both owned by the system, they are both referred to as internal ports. If all ports owned by a system are internal
then the system can in no way communicate with an external environment (and therefore is of little interest!); such a system is said to be a closed system.

**Definition 3.5.4** For every set \( Q \subseteq P \) we define \( \text{ext}(Q) = \{ p \in Q : \neg p \notin Q \} \).

For every system \( C \) we use \( \text{ext}(C) = \text{ext}(P(C)) \) to denote the set of the external ports of that system.

A system is said to be closed iff \( \text{ext}(C) \) is the empty set; otherwise \( C \) is open. □

We end this section by formulating a number of properties required for the subsequent development, the most important being that events preserve admissibility (Lemma 3.5.6(i)) and external connectivity (Lemma 3.5.7), these properties being a result of the syntactic restrictions \( A1...A8 \) imposed on the behaviour specifications which define possible events.

**Lemma 3.5.5** Let \( W,T \in \text{Adm} \).

(i) \( S \in \text{Adm} \), for all \( S \subseteq W \).

(ii) If \( P(W) \cap P(T) = \emptyset \) then \( W \cap T = \emptyset \) and \( W \cup T \in \text{Adm} \).

**Proof**

Follows directly from Definition 3.5.1 and 3.5.2. We only observe that \( W \cap T = \emptyset \) follows from the fact that each agent possesses a generator; and each message possesses at least one output. □

**Lemma 3.5.6** Let \( A \) be a behaviour specification as in Definition 3.3.1 or 3.4.1, and take any \((U,V) \in \Psi(A)\) such that \( U \in \text{Adm} \). Then the following are satisfied:

(i) \( V \in \text{Adm} \);

(ii) \( P(V) \subseteq P(U) \);

(iii) \( |V| = m+n \).

(Note that (iii) means that \((U,V)\) represents an event in which \( m \) new agents are created and \( n \) output messages are sent.)

**Proof**

Let \( A_c \) be a communication behaviour specification as in Definition 3.3.1, and let \( f \) be a substitution function as in Definition 3.3.2. Denote \( \text{var} = \{ \text{in}_0 \} \cup \{ \text{out}_0 \} \cup \{ \text{in}_{0,\rho} \} \cup \{ \text{out}_{0,\rho} \} \).

By the admissibility of \( U \) we obtain:

1. \( q/f \neq r/f \) for all \( q \neq r \in \text{var} \);
2. \( q/f \notin \{ (g_0.i)/f, -(g_0.i)/f \} \) for all \( q \in \text{var} \) and \( i \geq 1 \).

Also, by the definition of \([i:k]\) we have:

3. \([i:k](g_0)/f \) and \([j:k](g_0)/f \) have no ports in common for all \( i \neq j \geq 1 \);
4. all ports in \([j:k](g_0)/f \) are ports in \( g_0/f \) distinct from:

\[ (g_0.1)/f, -(g_0.1)/f, ..., (g_0.k)/f, -(g_0.k)/f. \]

We observe that if \( v \in V \) then \( [v] \) is an admissible (singleton) set such that \( P([v]) \subseteq P(U) \).

For if \( v \) is an agent then this follows from A.3, A.4, (1), (2) and (4); and if \( v \) is a message then this follows from A.3, (1) and (2).
Consequently, by Lemma 3.5.5(ii), to show (i), (ii), (iii) it is sufficient to demonstrate that $P(\mu)\cap P(\omega) = \emptyset$ for all $\mu \neq \omega \in V$. This, however, follows from A.3, A.4, (1), (2), (3) and (4). Proceeding similarly we show the thesis for a spontaneous behaviour specification. □

Lemma 3.5.7 Let $A$ be a behaviour specification, and let $(U,V) \in \Psi(A)$. If $U$ is admissible, then $ext(P(V)) = ext(P(U))$.

Proof
Similar to that of Lemma 3.5.6 (the conditions A.3 and A.7 are crucial). □

3.6 Operational semantics of systems

The possible evolutions of a system are given by a transition relation $\rightarrow$ which includes the internal events prescribed by a system's behaviour specifications and all possible interactions with an external environment. There are three different kinds of transition described below, each transition transforming system $C$ into a configuration $D$ (which it will be shown satisfies the requirements for being a valid system). These are illustrated in Figure 2, where a labelled boundary, e.g. $X$, is used to enclose the objects comprising system $X$, and links crossing the boundaries are external links of $X$.

![Figure 2](image)

An arrival transition, denoted $C \rightarrow_{+\mu} D$, is a message $\mu$ from the external environment becoming part of $C$, and thus available to participate in an event. We allow the possibility of an arrival transition for any message with its destination within $W_C$ and which when combined with $W_C$ gives an admissible set of object. (The first transition of Figure 2, $X \rightarrow_{+\mu} X_1$, is an arrival
transition for message \( \mu \).)

An internal transition, denoted \( C \rightarrow_w D \), is an event \( w \) given by applying a (spontaneous or communication) specification in \( \Gamma_C \) to an agent \( \alpha \) and possibly a message \( \mu \) in \( W_C \) to produce a system \( D \) in which \( \alpha \) and \( \mu \) have been replaced by the successor agents and output messages defined by the event. (The second transition of Figure 2, \( X_1 \rightarrow_w X_2 \), is an internal transition in which message \( \mu \) is received by agent \( \alpha \), giving rise to a new agent \( \beta \) and output message \( \mu_1 \).)

To define the possible events for a system we must extend the definition of \( \Psi(A) \) of the possible events prescribed by a single behaviour specification \( A \), to give \( \Psi(\Gamma) \) for a set of behaviour specifications \( \Gamma \) (a system's script) as the set of all possible events prescribed by any of its constituent behaviour specification, That is, for any script \( \Gamma \), \( \Psi(\Gamma) = \{ w : \exists A \in \Gamma : w \in \Psi(A) \} \).

A departure transition, denoted \( C \rightarrow_{-\mu} D \), is a message \( \mu \) disappearing from \( C \) and thus becoming part of the external environment. We allow the departure of any message with its destination outside \( W_C \). (The third transition of Figure 2, \( X_2 \rightarrow_{-\mu_1} X_3 \), is a departure transition for message \( \mu_1 \).)

**Definition 3.6.1** Let \( Sys \) and \( Conf \) denote the sets of all systems and configurations, respectively.

(i) For every \( \mu \in Mess \) we use \( \rightarrow_{+\mu} \) to denote a relation \( \rightarrow_{+\mu} \subseteq Sys \times Conf \) such that:

\( C \rightarrow_{+\mu} D \) iff: \( dest(\mu) \in ext(C) ; (W_C \cup \{ \mu \}) \in Adm ; \ W_D = (W_C \cup \{ \mu \}) \) and \( \Gamma_D = \Gamma_C \).

(ii) For every \( w = (U,V) \in (Ag \cup Mess) \times (Ag \cup Mess) \), \( \rightarrow_w \subseteq Sys \times Conf \) is a relation such that:

\( C \rightarrow_w D \) iff: \( w \in \Psi(\Gamma_C) ; U \subseteq W_C ; W_D = ((W_C \cup U) \cup V) \) and \( \Gamma_D = \Gamma_C \).

(iii) For every \( \mu \in Mess \) we use \( \rightarrow_{-\mu} \) to denote a relation \( \rightarrow_{-\mu} \subseteq Sys \times Conf \) such that:

\( C \rightarrow_{-\mu} D \) iff: \( \mu \in W_C ; dest(\mu) \in ext(C) ; W_D = (W_C - \{ \mu \}) \) and \( \Gamma_D = \Gamma_C \).

We denote \( C \rightarrow D \) if one of the above transitions is satisfied by a system \( C \) and a configuration \( D \). Also if \( C \rightarrow_{w_1} C_1 \rightarrow_{w_2} C_2 \ldots \rightarrow_{w_k} D \) we say \( s = (w_1,\ldots,w_k) \) is an event sequence for \( C \) giving \( D \); denoted \( C \rightarrow D \).

The transition relation defined here satisfies two properties (Theorems 3.6.4 and 3.6.5 below):

(a) due to the syntactic restrictions imposed on behaviour specifications above, transitions preserve admissibility and thus the non-sharing of ports; that is if \( C \rightarrow D \) then \( D \) is a system; and

(b) internal transitions do not change the external connectivity of a system; that is \( C \rightarrow_w D \) implies \( ext(D) = ext(C) \). (However, arrival and departure transitions do, necessarily, change the external connectivity.) Having introduced the transition relation, \( \rightarrow \), the definition of an operational semantics of a system is quite straightforward.

**Definition 3.6.2** With every system \( C \) we associate the set of all its possible computations, a computation being any (finite or infinite) sequence \( C_1, C_2, \ldots \) of systems such that \( C_1 = C \) and \( C_{i+1} \rightarrow C_i \) for all \( C_i \) with \( i \geq 2 \).
An important feature of a model dealing with distributed systems is its ability to formulate the concept of "fairness". In this model we can express a "fairness" property, for instance by defining a computation as being unfair if at some stage in the computation there is some event which thereafter always can occur but never actually does.

**Definition 3.6.3** An infinite computation \( C_1, C_2, \ldots \) is said to be **unfair** iff there is \((U, V) \in \Psi(T_{C_i})\) and \( k \geq 1 \) such that \( U \subseteq W_{C_i} \) for all \( i \geq k \). Otherwise it is **fair**. □

![Diagram of a computation, a "precedes" relation, and an induced partial order.](image)

**Figure 3**

A system's computation in its direct (sequential) form gives a total ordering of events, whereas in reality there is only a partial order. The underlying partial ordering on the computation's events can be recovered from the objects involved in those events. Intuitively, if an object was created in event \( E \) and destroyed in event \( E' \), then \( E \) occurred before \( E' \). Suppose that \( C_1, \ldots, C_{k+1} \) is a computation and \( C_i \rightarrow_{w_i} C_{i+1} \) for \( i = 1, \ldots, k \), where \( w_i = (U_i, V_i) \). We
associate with this computation a partial order \( (w_1, \ldots, w_k, \Rightarrow) \), where \( \Rightarrow \) is the transitive reflexive closure of a "precedes" relation \( \leadsto \) on the events \( w_i \), defined as \( w_i \leadsto w_j \) iff \( V_i \cap U_j \neq \emptyset \) for all \( i, j \leq k \). In Figure 3 we graphically represent the way in which we construct this partial order indicating for each arc between events in the \( \leadsto \) relation the common object which establishes the ordering of those events. (Note that the validity of the \( \leadsto \) relation depends on the impossibility of two distinct events creating an identical object.)

Finally we prove the two important properties that the transition relation on systems preserves admissibility and external connectivuity.

**Theorem 3.6.4** If \( C \in \text{Syst} \), \( D \in \text{Conf} \) and \( C \rightarrow D \) then \( D \in \text{Syst} \).

**Proof**

We consider three cases.

**Case 1:** \( C \rightarrow_+ \mu \) \( D \). Follows directly from Definition 3.6.1(i).

**Case 2:** \( C \rightarrow_\omega \) \( D \), where \( w = (U, V) \in \Psi(T_C) \). We have \( W_D = (W_C \cup U) \). By Lemma 3.5.5(i), \( (W_C \cup U) \) is an admissible set. Clearly, \( P(W_C \cup U) \subseteq P(W_C \cup P(U)) \). On the other hand, by Lemma 3.5.6, \( V \in \text{Adm} \) and \( P(V) \subseteq P(U) \). Hence, by Lemma 3.5.5(ii), we have \( W_D = (W_C \cup U) \in \text{Adm} \).

**Case 3:** \( C \rightarrow_\mu \) \( D \). Follows from Lemma 3.5.5(i). \( \square \)

**Theorem 3.6.5** If \( C, D \in \text{Syst} \) and \( C \rightarrow_\omega \) \( D \), where \( w \in \Psi(T_C) \), then \( \text{ext}(C) = \text{ext}(D) \).

**Proof**

Let \( W = W_C \) and let \( w = (U, V) \). By Lemma 3.5.7, \( \text{ext}(P(U)) = \text{ext}(P(V)) \). Thus:

\[
\text{ext}(D) = \text{ext}(P(W \cup U)) = \text{ext}(\text{ext}(P(W \cup U)) \cup \text{ext}(P(V))) = \text{ext}(\text{ext}(P(W \cup U)) \cup \text{ext}(P(U)))
\]

\[
= \text{ext}(P(W \cup U)) = \text{ext}(P(W)) = \text{ext}(C). \quad \square
\]

### 3.7 System Composition

We now define the operation of combining two systems as one, for which we use an operator \( \oplus \). For two systems \( C \) and \( D \) the composition \( (C \oplus D) \) is defined as being the system comprising the initial objects in both \( C \) and \( D \), and the behaviour specifications in both \( C \) and \( D \). The \( \oplus \) operator is only defined for a pair of systems \( C, D \) which are disjoint, by which we mean that: (a) \( C \) and \( D \) have no ports in common (since otherwise the combined system may not preserve the port privacy property); and (b) no behaviour identifier occurring in \( C \) also occurs in \( D \), and vice-versa (since otherwise in the combined system \( (C \oplus D) \), a behaviour specification coming from \( D \) could affect the possible behaviour of an agent coming from \( C \)).

**Definition 3.7.1** Two systems \( C \) and \( D \) are said to be disjoint iff: \( P(C) \cap P(D) = \emptyset \) and \( ID(C) \cap ID(D) = \emptyset \), where for every system \( E \), \( ID(E) = ID(W_E) \cup ID(T_E) \); \( ID(W_E) \) denoting the
set of all behaviour identifiers in agents of $W_E$; and $ID(T_E)$ denoting the set of all behaviour identifiers used by the agent expressions in specifications of $T_E$. □

**Definition 3.7.2** If $C$ and $D$ are two disjoint systems then their composition $(C \oplus D)$ is defined by: $(C \oplus D) = \langle (W_C \cup W_D); (\Gamma_C \cup \Gamma_D) \rangle$. □

**Corollary 3.7.3** If $C$ and $D$ are two disjoint systems then their composition $(C \oplus D)$ is a system. □

We must make here an important remark. It will always be implicitly assumed that a system $C$ does not use “too many” ports and behaviour identifiers (i.e. $P \cdot P(C)$ is an infinite set and there is infinitely many behaviour identifiers which are not used in $C$); otherwise $C$ could not be composed with other non-trivial systems. It will be also assumed that renamings of ports and relabellings of behaviour identifiers (to be defined in the next section) that can be applied to a given system preserve this property.

In composing two systems, $S = C \oplus D$, there will, in general, be some port pairs $(p, \cdot p)$ such that $p$ is an external outport of $C$ and $\cdot p$ is an external import of $D$ (or vice-versa). It is those ports which give the common links enabling interaction between the two composed systems, and in the combined system those ports will be internal ports. This is illustrated in Figure 4 for the (“successful”) combination of system $C$ and $D$. In the possible computations for $C$ (Figure 4(a))
there may be an internal transition \( T_1 \) in which a message with outport \( p \) as destination is created, followed by a departure transition \( T_2 \) in which that message is made available to \( C \)'s external environment. In the possible computations for \( D \) (Figure 4(b)) there may be an arrival transition \( T_3 \) in which a message with \( p \) as the destination outport is accepted from \( D \)'s external environment followed by an internal transition \( T_4 \) in which that message is received by an agent of \( D \). In the possible computations for the combination of \( C \) and \( D \) (Figure 4(c)), departure and arrival transitions \( T_2 \) and \( T_3 \) will not occur since \( p \) is now an internal port.

![Diagram](image)

**Figure 5**

*Figure 5* shows a different example in which the combination of \( C \) and \( D \) gives deadlock. In the possible computations for \( C \) (Figure 5(a)) there may be a transition \( T_0 \), the arrival of an external message for \( C \), with \( q \) as the destination outport, which is then received in transition \( T_1 \). Also, in the possible computations for \( D \) (Figure 5(b)) there may be a departure transition \( T_5 \) of a message with the destination outport \( q \) which has been created in transition \( T_4 \). Clearly, in the composed system \( C \oplus D \) (Figure 5(c)) none of the transitions \( T_0-T_5 \) may happen. The elimination of those transitions may of course also eliminate other transitions which were possible for \( C \) and \( D \) in isolation. Thus although the composition of two systems is a simple union of their objects and specifications, the set of possible computations for the combined system is not necessarily a simple function of the possible computations for both systems in isolation.
3.8 Port Renaming and Behaviour Identifier Relabelling

The above defined composition of systems is only valid for systems which are disjoint. In order to allow composition of arbitrary systems $C$ and $D$ we introduce port renamings and behaviour identifier relabellings which can be applied to say $D$ to produce a system which is disjoint with $C$. A port renaming is a one-to-one mapping which preserves port connectivity and port quality; and a relabelling of behaviour identifiers is a one-to-one mapping with the domain and range being the set of behaviour identifiers.

**Definition 3.8.1** A mapping $h : P \rightarrow P$ is called a *renaming of ports* iff: $h(P_{OUT}) = P_{OUT}$; $h(P_{IN}) = P_{IN}$; $h(p) \neq h(q)$; and $h(p) = -h(p')$, for all $p \neq q \in P$.

When applied to a system $C$ a renaming $h$ yields a new system $C[h]$ where all ports $p$ are replaced by $h(p)$. That is, if $a = (\gamma; g; in; out; x)$ is an agent, then by $a[h]$ we denote an agent $a[h] = (\gamma; h(g); in'; out'; x)$; where $(g'; i) = h(g; i)$ for all $i$; $in'= (h(p_1),...,h(p_n))$ for $in = (p_1,...,p_n)$ and $out' = (h(q_1),...,h(q_n))$ for $out = (q_1,...,q_n)$. Similarly, we define $\mu[h]$ for every message $\mu$ and for a configuration $C$ we use $C[h]$ to denote the configuration $C[h] = <W_C[h],\Gamma>$, where for every set $W \subseteq (Ag \cup Mess)$, $W[h] = \{w[h] : w \in W\}$.

**Definition 4.8.2** A mapping $\pi : ID \rightarrow ID$ is a *relabelling of behaviour identifiers* iff $\pi(ID) = ID$ and $\pi(\gamma) \neq \pi(\omega)$, for all $\gamma \neq \omega \in ID$.

We can apply a relabelling $\pi$ to any system $C$, to give a new system $\pi(C)$ where all behaviour identifiers $\gamma$ are replaced by $\pi(\gamma)$. That is, if $\mu$ is a message expression, then $\pi(\mu) = \mu$; and if $a = (\gamma; g; in; out; x)$ is an agent expression, then $\pi(a) = (\pi(\gamma); g; in; out; x)$. Then, in a natural way we define $\pi(A)$, for any behaviour specification $A$. Finally, we define $\pi(C) = <\pi(W_C); \pi(\Gamma_C)>$, where $\pi(W_C) = \{\pi(w) : w \in W\}$ and $\pi(\Gamma_C) = \{\pi(A) : A \in \Gamma_C\}$.

The renaming of ports and relabelling of behaviour identifiers in a system does not change its structure nor can it in any essential way change its behaviour. In particular, we have the following.

**Corollary 3.8.3** Let $C$ be a system.

(i) If $h$ is a renaming of ports then $C[h]$ is a system.

(ii) If $\pi$ is a relabelling of behaviour identifiers then $\pi(C)$ is a system.

**Corollary 3.8.4** Let $C \rightarrow^\omega D$ for a system $C$ and some $\omega = (U,V) \in \Psi(\Gamma_C)$.

(i) If $h$ is a renaming of ports then $C[h] \rightarrow^w_D[h]$, where $w(h) = (U(h),V(h))$.

(ii) If $\pi$ is a relabelling of behaviour identifiers then $\pi(C) \rightarrow^{\pi(\omega)} \pi(D)$, where $\pi(\omega) = (\pi(U),\pi(V))$. 

-20-
4 OBSERVATIONAL ASPECTS OF SYSTEMS BEHAVIOUR

The observational aspects of a system's behaviour are those which determine how the system may interact with, and thus influence, its external "observing" environment. Our main objective here is to make precise what it means for two systems to be observationally equivalent. To illustrate the issues that arise we will consider an example shown in Figure 6.

In the first stage of Figure 6(a), there is a system, $G_0$, composed of two component systems, $O_0$ (the observer) and $C_0$ (the observed); and in the first stage of Figure 6(b) there is a system, $H_0$, composed of the same observer and a different observed, $D_0$; the issue being observational equivalence between $C_0$ and $D_0$. The systems $G_1, G_2, G_3$ are successive systems of one possible computation for $G_0$, with event sequence $s = (w_1, w_2, w_3)$; and the systems $H_1, H, H_2, H_3$ with event sequence $t = (w_1, v, v_1, w'_3)$, being a corresponding possible computation for $H_0$. (The use of the same identifying symbol for an entity (event, message, agent, link, system) in both (a) and (b) means the identified entities are identical in both cases; use of primed and un-primed identifying symbols for two non-identical identities, e.g. $\mu_2'$ in (b) and $\mu_2$ in (a), means that the two entities are in correspondence in a sense to be subsequently defined.)

The system in (a) is similar to the User-Manager-File system of Figure 1. In event $w_1$, the user agent ($U_0$) sends a message ($\mu_1$) to the manager agent ($M_0$) which in event $w_2$ forwards that message as $\mu_2$ to the file agent ($F_0$) where it is received in event $w_3$. Here (unlike Figure 1) the file agent ($F_0$) creates a separate agent ($V_0$) to service the user's access to the file (now agent $F_1$), with bi-directional communication between the service and file agents. We are assuming that for the initial agents $U_0, M_0$ and $F_0$ there are port generators: $(u_{1\text{outport}}, u_{2\text{outport}}, \ldots), (m_{1\text{outport}}, m_{2\text{outport}}, \ldots), (f_{1\text{outport}}, f_{2\text{outport}}, \ldots)$, and have labeled accordingly the links $u_{1}, u_{2}$ etc., created by those agents and their successors.

The system in (b) is the same as that in (a) except that the manager agent ($N_0$) does not allow a direct connection between the user and the file (link $u_{2}$ in $G_3$). Instead, a "censor" agent $X$ is created and interposed between the user and the file; compare: event $w_2$ of (a) where the message $\mu_1$ is directly forwarded as $\mu_2$; with events $v$ and $v_1$ of (b) which perform the interposition of the censor $X$. In (a), messages that $U_1$ and its successor agents send on link $u_{2}$ and its successor links will be directly receivable by $V_0$ and its successor agents. In contrast, such messages in (b) will be received by the censor agent $X$ and its successors which can decide whether to forward a message essentially unchanged, block it, or forward it in a modified form. The important point of this example is that if we have a censor agent which is totally "permissive", that is if it (and its successors) forward all possible messages essentially unchanged (i.e. without changing structure or data values); then the interposition of this censor is undetectable by any
observing system such as \( O_0 \); and we should have a definition which gives \( C_0 \) and \( D_0 \) as being observationally equivalent.

The usual approach to observational equivalence would proceed by defining for \( C_0 \) and \( D_0 \) a characterisation of their possible external interactions, and defining those systems as observationally equivalent if there is equivalence between those characterisations. In adopting this approach for our model the external interaction characterisations would be in terms of the possible arrival and departure transitions for the systems, that is, what messages could arrive from and be given up to the external environment. The inadequacy of the direct application of this approach is seen at the point where \( C_0 \) has evolved to \( C_2 \) and \( D_0 \) to \( D_2 \): there are for \( D_2 \)
two possible message arrivals (a message on link \( u2 \) and a message on link \( u1 \)); whereas for \( C_2 \) only the latter message arrival is possible. Differences such as these would produce differences in any straightforward formulation of the external interaction characterisations of \( C_0 \) and \( D_0 \), and thus their observational non-equivalence. The essential difficulty is that the dynamic structuring of our model allows the external connections of a system to change. This difficulty of course does not arise in statically structured models such as CSP and CCS and the usual "external interactions" approach is adequate for the more restricted class of systems to which those models are applicable. The dynamically structured ACTORS model does allow the external connections of a system to change, but avoids the issues raised by examples such as this by giving an "external interactions" definition of observational equivalence (AGH) which would result in a pair of systems such as \( C_0 \) and \( D_0 \) being classified as non-observationally-equivalent.

The approach which we will adopt here is to define observational equivalence in terms of the behaviour of the observing system rather than that of the observed system. We thus sidestep the problem identified above, and also have a definition which seems to us to more directly capture what is intuitively meant by observational equivalence. (This definition does result in \( C_0 \) and \( D_0 \) being observationally equivalent, an outline proof of this being given in the appendix.) The essence of our definition of observational equivalence of two systems such as \( C_0 \) and \( D_0 \) is the requirement that if we take any observer such as \( O_0 \), then, for any possible event sequence, \( s \), for \( G_0 = (O_0 \oplus C_0) \), there will be a corresponding event sequence \( t \) for \( H_0 = (O_0 \oplus D_0) \). For example, \( s = (w_1, w_2, w_3) \) and \( t = (w_1, v, v_1, w_3) \) are such corresponding sequences and we now define precisely what is meant by that correspondence.

4.1 Correspondence of Event Sequences

In defining two event sequences \( s \) and \( t \) as corresponding, we require that the correspondence depend only on certain relevant events, namely those events which occur within the observing parts of the two systems, that is: for system \( G_0 \), events \( w_1 \) and \( w_2 \) are relevant and \( w_2 \) is irrelevant; and for \( H_0 \), events \( w_1 \) and \( w_3 \) are relevant and \( v \) and \( v_1 \) are irrelevant. It is convenient to characterise the relevant, observing, part of the system \( G_0 \) (or \( H_0 \)) and its successors \( G_1 \) etc. (or \( H_1 \) etc.) by the script, say \( \Delta \), of the \( O_0 \) component system since this remains constant as the system evolves. Any relevant event will be one of those prescribed by \( \Delta \), whereas any irrelevant event will be one of those prescribed by the script for the \( C_0 \) or \( D_0 \) component. (Note, the validity of this characterisation depends on the disjointness of the scripts for \( O_0 \) and \( C_0 \) (or \( D_0 \)) which is necessary for their composition as \( G_0 \) (or \( H_0 \)) to be formed.)

To formally capture the requirement that correspondence between event sequences depend only on the relevant events characterised by the script \( \Delta \), we define a projection function \( \Delta \) which maps a sequence of events such as \( s = (w_1, w_2, w_3) \) into the relevant sequence \( (w_1, w_3) \).
Definition 4.1.1 Let $\Gamma$ and $\Delta$ be two scripts. For every sequence $s = (w_1, ..., w_k) \in \Psi(\Gamma)^*$ we define $s|_\Delta = (w_{i_1}, ..., w_{i_m})$, where:

$m \geq 0$, $1 \leq i_1 < ... < i_m \leq k$ and for all $j \leq k$, $w_j \in \Psi(\Delta) \Leftrightarrow j \in \{i_1, ..., i_m\}$. □

We will now define two event sequences such as $s$ (for $G_0$) and $t$ (for $H_0$) as corresponding, $s \sim_\Delta t$, with respect to script $\Delta$ characterising relevant events, if there is a correspondence, $s_i \sim_\Delta t_i$, between events $s_i$ and $t_i$ at the same positions in the two projected sequences $s|_\Delta (= (w_1, w_2))$ and $t|_\Delta (= (w_1, w_3))$. For any pair of events such as $w_2$ and $w_2$ to be considered corresponding we require the following properties:

1 Considering two objects to "correspond" if they are identical except for possible differences in the ports for their links with other objects, then:

1a the principle agents of the events correspond and so do the input messages;

1b for every successor agent or output message of one event there is a corresponding successor agent or output message of the other event (and thus there are the same possibilities in both systems for subsequent corresponding events involving those corresponding successor agents).

The four objects involved in event $w_3$ ($F_0$ - principle agent, $\mu_2$ - input message, $V_0$ and $F_1$ - successor agents), and the corresponding objects for $w_3$ are identical except that port $w_2$ in the former appears as $m_1$ in the latter and port $m_1$ in the former appears as $m_2$ in the latter. The differences in the identities of ports are undetectable in that the syntax for behaviour specifications does not allow an agent to test or derive a value from the identity of a port - if we did not allow such differences between "corresponding" entities, then the eventual definition of observational equivalence would be too strong, whereas if we allowed greater differences, e.g. in data values, it would be too weak.

2 Both events can be derived as applications of the same behaviour specification $A$ from $\Delta$ (the observing system's script) although in the two cases there may be differences in the assignment of port values to port variables of $A$. Properties 1a and 2 are sufficient to derive property 1b (Corollary 3.3.3 and 3.4.3), and for convenience we define correspondence of events, and thus event sequences, using just the former.

Definition 4.1.2 Let $\Delta$ be a script, and let $w, v \in \Psi(\Delta)$, where $w = (U, V)$ and $v = (U', V')$.

We say that $w$ and $v$ are corresponding events, $w \sim_\Delta v$, iff there is $A \in \Delta$ such that $w, v \in \Psi(A)$ and:

(i) If $w, v$ are spontaneous events then $g = g'$ and $x = x'$,
where $U = \langle (y; g; \text{in}; \text{out}; x) \rangle$ and $U' = \langle (y; g'; \text{in'}; \text{out'}; x') \rangle$;

(ii) If $w, v$ are communication events then $g = g'$, $x = x'$ and $x_0 = x'_0$,
where $U = \langle (y; g; \text{in}; \text{out}; x), (\text{in}_0; \text{out}_0; x_0) \rangle$ and $U' = \langle (y; g'; \text{in'}; \text{out'}; x'), (\text{in'}_0; \text{out'}_0; x'_0) \rangle$. □
Definition 4.1.3 Let $\Delta, \Gamma$ be two scripts and let $s, t \in \Psi(\Gamma)^*$. We say that $s$ and $t$ are corresponding event sequences, $s \sim_{\Delta} t$, iff $\#(s|_{\Delta}) = \#(t|_{\Delta})$ and we have $s_i \sim_{\Delta} t_i$ for $i = 1, \ldots, k$, where $s|_{\Delta} = (s_1, \ldots, s_k)$ and $t|_{\Delta} = (t_1, \ldots, t_k)$. 

4.2 Partial Operational Equivalence

In preparation for finally defining observational equivalence in the next subsection, we now define a notion of Partial Operational Equivalence (POE) between systems (of a certain restricted class) which captures the sense in which $G_0$ and $H_0$ of Figure 6 are equivalent with respect to the script $\Delta$ which characterises the relevant parts of those systems, this equivalence being denoted $G_0 \approx_{\Delta} H_0$. First we define the class of systems, to which a POE $\approx_{\Delta}$ is applicable. In this we impose the following two restrictions on a system $G$:

1. We require $G$ to be a closed system since this simplifies the development and is without loss of generality in the eventual definition of observational equivalence.

2. We require the property that in the complete script $\Gamma_G$ for $G$ there be disjointness between the behaviour specifications $\Delta$, and the remaining behaviour specifications of $\Gamma_G$. This property, $\Delta$-disjointness, is required for the validity of the definition of relevant events for the equivalence and will hold for any system which is (or has evolved from a system which is) a composition of two systems, such that $\Delta$ is the script for one of the components.

Definition 4.2.1 Let $\Delta$ be a script and $G$ be a system. Then $G$ is said to be $\Delta$-disjoint iff $ID(\Delta) \cap ID(\Gamma_G - \Delta) = \emptyset$. 

We now can define the notion of partial operational equivalence of two systems. We adopt here the concept of a bisimulation relation introduced in [PAR].

Definition 4.2.2 Let $\Delta$ be a script. A binary relation $\Sigma$ on closed $\Delta$-disjoint systems is defined to be a $\Delta$-bisimulation iff for each pair $(G,H) \in \Sigma$ and for every $s \in \Psi(\Gamma_G \cup \Gamma_H)^*$ we have:

- if $G \rightarrow_s G_0$ then there are $t,H_0$ such that: $H \rightarrow t,H_0$, $t \sim_{\Delta} s$ and $(G_0,H_0) \in \Sigma$;
- if $H \rightarrow_s H_0$ then there are $t,G_0$ such that: $G \rightarrow t,G_0$, $t \sim_{\Delta} s$ and $(G_0,H_0) \in \Sigma$. 

(Note that for every computation for $G$ there is a corresponding computation for $H$, and vice versa, and that the same holds for the final systems $G_0$ and $H_0$ of those computations. In some sense $H$ "simulates" $G$, and $G$ also simulates $H$.)

Definition 4.2.3 Two closed $\Delta$-disjoint systems $G$ and $H$ are said to be partial-operationally equivalent with respect to a script $\Delta$ iff there is a $\Delta$-bisimulation $\Sigma$ such that $(G,H) \in \Sigma$. We denote this by $G \approx_{\Delta} H$. 

We now prove a number of lemmas required subsequently, the most important being that POE is an equivalence relation in the usual technical sense, and that it is preserved by renaming of ports and relabelling of behaviour identifiers (Lemma 4.2.5 and 4.2.6).
Lemma 4.2.4 For every script $\Delta$ the following are satisfied:

(i) $\approx_\Delta$ is a $\Delta$-bisimulation;

(ii) $\approx_\Delta$ is an equivalence relation.

Proof

(i) Follows directly from Definition 4.2.3.

(ii) Clearly, $\approx_\Delta$ is reflexive since the relation $\langle(G,G) : G$ is a closed $\Delta$-disjoint system$\rangle$ is a $\Delta$-bisimulation; and $\approx_\Delta$ is symmetric since for every $\Delta$-bisimulation $\Sigma$ the relation $\langle(H,G) : (G,H)\in\Sigma\rangle$ is also a $\Delta$-bisimulation.

To show the transitivity of $\approx_\Delta$ it suffices to demonstrate that for any two $\Delta$-bisimulations $\Sigma_1$ and $\Sigma_2$ their composition

$$\Sigma = \langle(G,H) : \exists F : (G,F)\in\Sigma_1 \land (F,H)\in\Sigma_2\rangle$$

is a $\Delta$-bisimulation.

Let $(G,H)\in\Sigma$. Then, for some $F$, we have $(G,F)\in\Sigma_1$ and $(F,H)\in\Sigma_2$.

Suppose that $G\rightarrow_s G_0$. Then, by $(G,F)\in\Sigma_1$, there are $z,F_0$ such that $F\rightarrow_z F_0$, $s\sim_\Delta z$ and $(G_0,F_0)\in\Sigma_1$. Now, by $(F,H)\in\Sigma_2$, there are $t,H_0$ such that $H\rightarrow_t H_0$, $t\sim_\Delta z$ and $(F_0,H_0)\in\Sigma_2$. Consequently, since $\sim_\Delta$ is transitive, $t\sim_\Delta s$ and $(G_0,H_0)\in\Sigma$.

Proceeding similarly we show that if $H\rightarrow_s H_0$ then we can find $t,G_0$ such that $G\rightarrow_t G_0$, $t\sim_\Delta s$ and $(G_0,H_0)\in\Sigma$.

This completes the proof. □

Lemma 4.2.5 If $\Delta$ is a script and $h$ is a renaming of ports, then for all closed $\Delta$-disjoint systems $G$ and $H$,

$$G\approx_\Delta H \Rightarrow G[h]\approx_\Delta H[h].$$

Proof

It suffices to show that $\Sigma = \langle(E[h],F[h]) : E =_\Delta F\rangle$ is a $\Delta$-bisimulation.

Let $(E[h],F[h])\in\Sigma$. Clearly, $E[h]$ and $F[h]$ are $\Delta$-disjoint systems. Suppose now that $E[h]\rightarrow_s J$, where $s\in\Psi(T_\Delta)^+$. By Corollary 3.8.4(i), $E[h][h^{-1}]\rightarrow_{\eta[h^{-1}]} J[h^{-1}]$, where $h^{-1}$ denotes the inverse of $h$ (which, of course, is itself a renaming), and $\eta[h^{-1}]$ is defined as $(s_1[h^{-1}],...,s_k[h^{-1}])$ for $s = (s_1,...,s_k)$. Now, since $E[h][h^{-1}] = E$ and $E =_\Delta F$, there are $y,F_o$ such that $F\rightarrow_y F_o$, $y\sim_\Delta s[h^{-1}]$ and $J[h^{-1}] =_\Delta F_o$. Then, again by Corollary 3.8.4, $F[h]\rightarrow_{\eta[h]} F_o[h]$. Clearly, since $y\sim_\Delta s[h^{-1}]$, we have $y[h]\sim_\Delta s[h^{-1}][h]$, and so $y[h]\sim_\Delta s$. On the other hand, since $J[h^{-1}] =_\Delta F_o$, we have $(J[h^{-1}][h], F_o[h])\in\Sigma$. Hence $(J,F_o[h])\in\Sigma$, which completes the proof. □

Lemma 4.2.6 Let $\Delta$ be a script and let $\pi$ be a relabelling of behaviour identifiers.

(i) If $G$ is a closed $\Delta$-disjoint system and $\pi(\gamma) = \gamma$ for all $\gamma \in ID(\Delta)$, then $G\approx_\Delta \pi(G)$.

(ii) If $G\approx_\Delta H$ then $\pi(G)\approx_{\pi(\Delta)} \pi(H)$, for any two closed $\Delta$-disjoint systems $G$ and $H$.

Proof

(i) One can easily see that $\Sigma = \langle(H,\pi(H)) : H$ is a closed $\Delta$-disjoint system$\rangle$ is a
\(\Delta\)-bisimulation.

(ii) Here one can prove that \(\Sigma = \langle (\pi(G), \pi(H)) : G \equiv_\Delta H \rangle\) is a \(\pi(\Delta)\)-bisimulation. \(\square\)

**Lemma 4.2.7** Let \(\Delta\) be a script, and let \(h\) be a renaming of ports. Then \(G[h] \equiv_\Delta G\) for every closed \(\Delta\)-disjoint system \(G = <W; \Gamma>\) such that for every \(a \in (W \cap Ag_\Delta)\), \(a\) and \(a[h]\) have the same generators, where \(Ag_\Delta\) denotes the set of the agents with behaviour identifiers belonging to \(ID(\Delta)\).

**Proof**

It is sufficient to show that \(\Sigma\) defined as the set of all pairs \((H[h], H)\), where \(H\) is a closed \(\Delta\)-disjoint system and if \(a \in (W_H \cap Ag_\Delta)\) then \(a\) and \(a[h]\) have the same generators, is a \(\Delta\)-bisimulation. We first observe that \(H[h]\) is \(\Delta\)-disjoint.

Let \((H[h], H) \in \Sigma\).

Suppose that \(H \to\_w H_0\), where \(w \in \Psi(\Gamma_H)\). By Corollary 3.8.4, \(H[h] \to\_w [H[h], H_0]\). Clearly, if \(w \notin \Psi(\Delta)\) then \(w \sim_\Delta w[h]\). If \(w \in \Psi(\Delta)\) then we also have \(w \sim_\Delta w[h]\) since for all \(a \in (W_H \cap Ag_\Delta)\), \(a\) and \(a[h]\) have the same generators. Let \(w = (U, V)\). We observe that \((W_{H_0} \cap Ag_\Delta) = (W_H \cap Ag_\Delta) \cdot (U \cap Ag_\Delta) \cup (V \cap Ag_\Delta)\). Clearly, for all \(a \in (V \cap Ag_\Delta)\), \(a\) and \(a[h]\) have the same generators. Hence if \(a \in (W_{H_0} \cap Ag_\Delta)\) then \(a\) and \(a[h]\) have the same generators. Consequently, \((H_0, H) \in \Sigma\).

Proceeding similarly we may show that if \(H[h] \to\_v H'\), where \(v \in \Psi(\Gamma_h)\), then there are \(w, H_0\) such that \(H \to\_w H_0, w \sim_\Delta w\) and \((H', H_0) \in \Sigma\). (By Corollary 3.8.4, one can take \(v = w[h^{-1}]\) and \(H_0 = H[h^{-1}]\).)

Hence if \((E, F) \in \Sigma\) and \(E \to\_w E_0\) (or \(F \to\_w F_0\), where \(w \in \Psi(\Gamma_E) = \Psi(\Gamma_F)\), then there are \(v\) and \(F_0\) (or \(E_0\)) such that \(F \to\_v F_0\) (or \(E \to\_v E_0\), \(\sim_\Delta v\) and \((E_0, F_0) \in \Sigma\).

Thus \(\Sigma\) is a \(\Delta\)-bisimulation, which completes the proof. \(\square\)

### 4.3 Observational Equivalence

Having introduced the notion of partial operational equivalence of two closed systems, we can define two systems \(C\) and \(D\) as being observationally equivalent if for every observing system \(O\) that could be combined with both, the two combinations \(G = (C \oplus O)\) and \(H = (D \oplus O)\) are partially operationally equivalent with respect to the set of behaviour specifications of \(O\). We also make an important remark regarding the external ports of a system, namely observationally equivalent systems must have identical external imports and exports since otherwise they would allow different external interconnections.

**Definition 4.3.1** Two systems \(C\) and \(D\) for which \(\text{exit}(C) = \text{exit}(D)\) are said to be **observationally equivalent** iff \((C \oplus O) \equiv_\Delta (D \oplus O)\) for every system \(O = <W; \Delta>\) such that both \((C \oplus O)\) and \((D \oplus O)\) are defined and give closed systems. We denote this by \(C = D\). \(\square\)
In the Appendix we will give an example of a proof that two systems are observationally equivalent according to this definition. For that example we will use a generalisation of the systems $C_0$ and $D_0$ of Figure 6. The essentials of the proof is to first formally define a scheme by which a system such as $G_3$, can be transformed to a system such as $H_3$, by interposing a censor agent (or, in general, a chain of censor agents) on any link and renaming some ports; then construct the relation comprising the set of pairs $(G, H)$, where $H$ is derived from $G$ by that transformation; and finally show that this relation is a bisimulation.

We now formulate some important theorems concerning the above observational equivalence, namely - observational equivalence is preserved by renaming of ports and relabelling of behaviour identifiers; it is an equivalence relation, in the usual technical sense; it is abstract, i.e. it is preserved through system composition. First we will show that renaming of ports and relabelling of behaviour identifiers preserve observational equivalence.

**Theorem 4.3.2** Let $h$ be a renaming of ports and let $C, D$ be two systems.

(i) If $h(p) = p$ for all $p \in \text{ext}(C)$, then $C = C[h]$.  
(ii) If $C = D$ then $C[h] = D[h]$.

**Proof**

(i) Clearly, $\text{ext}(C) = \text{ext}(C[h])$.

Let $O = \langle W, \Delta >$ be a system such that $(C \oplus O)$ and $(C[h] \oplus O)$ are defined and give closed systems. Since $h(p) = p$ for all $p \in \text{ext}(C)$, we may find a renaming $d$ such that $C[d] = C[h]$ and $O[d] = O$. Hence, by Lemma 4.2.7, $(C \oplus O) \simeq \Delta (C \oplus O)[d]$. On the other hand, we have: 

$\Delta (C \oplus O)[d] = (C[d] \oplus O[d]) = (C[h] \oplus O)$.

Thus, $(C \oplus O) \simeq \Delta (C[h] \oplus O)$.

(ii) Clearly, $\text{ext}(C[h]) = \text{ext}(D[h])$.

Let $O = \langle W, \Delta >$ be a system such that $(C[h] \oplus O)$ and $(D[h] \oplus O)$ are defined and give closed systems. It is not difficult to see that $E = O[h^{-1}]$ is a system such that $(C \oplus E)$ and $(D \oplus E)$ are defined and give closed systems. We have $C = D$ and $\Gamma_E = \Delta$. This and Lemma 4.2.5 yields $(C \oplus E)[h] \simeq \Delta (D \oplus E)[h]$. Thus $(C[h] \oplus O) \simeq \Delta (D[h] \oplus O)$ which completes the proof.

**Theorem 4.3.3** Let $\pi$ be a relabelling of behaviour identifiers and let $C, D$ be two systems.

(i) $C = \pi(C)$.  
(ii) $C = D \Rightarrow \pi(C) = \pi(D)$.

**Proof**

Follows from Lemma 4.2.6.

**Theorem 4.3.4** $\simeq$ is an equivalence relation, i.e. if $C, D, E$ are systems then the following are satisfied:

(i) $C = C$,
(ii) \( C = D \iff D = C \)

(iii) \((C = D \land D = E) \Rightarrow C = E\).

**Proof**

(i), (ii) Follow from the fact that for any script \( \Delta \), \( \sim_\Delta \) is reflexive and symmetric (Lemma 4.2.4(ii)).

(iii) We first observe that \( \text{ext}(C) = \text{ext}(E) \).

Let \( O = <W;\Delta> \) be a system such that \((C \oplus O)\) and \((E \oplus O)\) are defined and give closed systems.

There is a relabelling of behaviour identifiers \( \pi \) and a renaming of ports \( h \) such that \( O \) is disjoint with \( \pi(C)[h] = \pi(C), \pi(D)[h] \) and \( \pi(E)[h] = \pi(E) \).

By Theorem 4.3.2 and Theorem 4.3.3 we have \( \pi(C)[h] = \pi(D)[h] \) and \( \pi(D)[h] = \pi(E)[h] \). Hence \( \pi(C) = \pi(D)[h] \) and \( \pi(D)[h] = \pi(E) \). Consequently, \( (\pi(C) \oplus O) \sim_\Delta (\pi(D)[h] \oplus O) \) and \( (\pi(D)[h] \oplus O) \sim_\Delta (\pi(E) \oplus O) \). Hence, by Lemma 4.2.4(ii), \( (\pi(C) \oplus O) \sim_\Delta (\pi(E) \oplus O) \). On the other hand, by Theorem 4.3.3, \( (C \oplus O) \sim_\Delta (\pi(C) \oplus O) \) and \( (E \oplus O) \sim_\Delta (\pi(E) \oplus O) \).

Thus, \( (C \oplus O) \sim_\Delta (E \oplus O) \), which completes the proof. \( \square \)

Finally, we will prove that \( = \) is abstract, i.e. it is preserved through system composition.

**Theorem 4.3.5** If \( C_1, C_2, D_1, D_2 \) are systems satisfying \( C_1 = D_1 \) and \( C_2 = D_2 \) then:

\((C_1 \oplus C_2) = (D_1 \oplus D_2)\), provided that \((C_1 \oplus C_2)\) and \((D_1 \oplus D_2)\) are defined.

**Proof**

We first prove that if \( C = D \) and \( E \) is a system which is disjoint both with \( C \) and with \( D \) then \((C \oplus E) = (D \oplus E)\).

Suppose \( O = <W;\Delta> \) is a system such that \( C_\theta = ((C \oplus E) \oplus O) \) and \( D_\theta = ((D \oplus E) \oplus O) \) are defined and give closed systems. By \( C = D \), we have \( C_\theta \approx_{\Gamma_\theta \cup \Delta} D_\theta \). It thus suffices to show that \( \Sigma = \approx_{\Gamma_\theta \cup \Delta} \) is a \( \Delta \)-bisimulation.

Let \((G,H) \in \Sigma \). Clearly, \( G \) and \( H \) are \( \Delta \)-disjoint. Suppose \( G \rightarrow_\theta G_\theta \). There are \( t,H_\theta \) such that \( H \rightarrow_\theta H_\theta, s \sim_{\Gamma_\theta \cup \Delta} t \) and \((G_\theta,H_\theta) \in \Sigma \). Clearly, we also have \( s \sim_\Delta t \), which completes the first part of the proof.

There will be some renaming of ports \( h \) and a relabelling of behaviour identifiers \( \pi \) such that:

\( \pi(C_2)[h] \) is disjoint with \( D_1 \); and \( h(p) = p \) for all \( p \in \text{ext}(C_1) \cup \text{ext}(C_2) \).

By Theorem 4.3.2 and Theorem 4.3.3, \( \pi(C_1)[h] = C_1 \). Thus \( \pi(C_1)[h] = D_1 \). Clearly, \( \pi(C_1) \) is disjoint with \( \pi(C_2)[h] \). Thus, from the first part of our proof it follows that \( (\pi(C_1)[h] \oplus \pi(C_2)[h]) = (D_1 \oplus \pi(C_2)[h]) \). Similarly, \( \pi(C_2)[h] = C_2 = D_2 \), and \( \pi(C_2)[h] \oplus D_1 = (D_2 \oplus D_1) \). Consequently, \( (\pi(C_1)[h] \oplus \pi(C_2)[h]) = (D_1 \oplus D_2) \). Hence \( \pi(C_1 \oplus C_2)[h] = (D_1 \oplus D_2) \). This and Theorem 4.3.2 and 4.3.3 yields \((C_1 \oplus C_2) = (D_1 \oplus D_2)\), which completes the proof. \( \square \)
5 CONCLUSION

In this paper we have presented the DSCS model of structure and communication for distributed systems which can dynamically change their internal structure, and can dynamically acquire (and lose!) connections with their external environment. As shown in a brief survey of the other principle models, these are generally limited in their application by requiring a fixed system configuration. The only other model with dynamic structure is limited by employing agent-based rather than port-based communication. For the DSCS model we have given the syntax and operational semantics of system specifications and a formal definition of systems which allows a very simple formulation for the general construction of a system as a composition of other systems. From this we developed a definition of observational equivalence between systems, adopting a novel approach which concerns itself with the operation of an observer rather than that of the observed. This approach deals simply and effectively with certain situations arising in dynamically structured systems which are difficult to handle in the usual approach to observational equivalence; and gives a definition which more directly captures the intuitive notion of observational equivalence. In the appendix we employed this definition to prove observational equivalence between two systems of significant functionality.

The operational semantics and observational equivalence developed here are intended to provide the basis of the further developments necessary for the model to be a useful system development tool, which would include: introducing a formalism for specifying required properties of a system and verifying that the full behaviour specification of a system satisfies those properties; identifying equivalence-preserving transformations on specifications; and designing a programming language notation for specifying the behaviour of agents (the behaviour specification notation used here being intended for ease of formal manipulation rather than for ease of designing and understanding actual systems of significant size). An approach currently being explored in the further developments is an application of the graph grammars formalism (EHR). One area in which the basic model needs to be developed is to allow dynamicism in the behaviour specifications which govern a system's operation—that is to be able to model systems which receive and behave in accordance with behavioural specifications provided as messages from the external environment, as occurs in systems which include compilers and loaders.

6 ACKNOWLEDGMENTS

We would like to thank Brian Randell for his numerous comments on earlier versions of this report; and the Science and Engineering Research Council for their support of this work.
7 REFERENCES


Appendix - A Proof of Observational Equivalence

We here present an outline proof of the observational equivalence of systems $C_0$ and $D_0$ used in Figure 6 (as discussed in Section 4), using in the explanation of the proof the User $U$ and File $F$ agents as initial components of an example observing system. We assume that the observed agents, $M$ and $N$, allow the creation of multiple paths from $U$ to $F$, and thus that the single service agent, $V$ in $G_3$ of Figure 6, can be generalised to multiple service agents. After developing a proof for this system we will indicate how it can be easily extended to systems such as $EC_0$ and $ED_0$, shown in Figure 8, which allow multiple users and files.

In Figure 7(a) and 7(b) we show systems $G_{10}$ and $H_{10}$ which could be produced by further evolution from $G_3$ and $H_3$ of Figure 6(a) and (b). We show three messages $v$, $\kappa$ and $\tau$, in order to exemplify those classes of events that have to be considered in establishing observational equivalence of $C_0$ and $D_0$. The sequence $G_{10} \rightarrow_{u_9} G_{11} \ldots G_{13}$ of (a) shows the processing of these three messages in the evolution from $G_{10}$. The $v$ is a message on the link which in $G_3$ (Figure 6) was linking user agent $U_1$ to service agent $V_0$ ($U_1$ and $V_0$ having in $G_{10}$ become their successors $U$ and $V$). This message carries the import of a link so that when $V$ has received $v$ in event $u_9$, that link will allow a subsequent communication from $U$ to $V$'s successor $V_1$. The termination of a connection from $U$ to a service agent can be achieved by sending on it a "terminal" message which does not have an import and thus does not allow a subsequent communication. This is illustrated by the second message, $\kappa$, sent from $U$ to a different service agent, $W$ - on receipt of that message, event $u_7$, the successor service agent $W_1$ does not have an import on which it could receive a message from $U$. The third message, $\tau$, is of the type used to establish a new service agent giving a new access path from $U$ to file agent $F$. This message is received by $M$ which forwards it as $\tau_1$, as shown in event $w_9$. Subsequently $\tau_1$ would be received by $F$ in an event similar to event $w_3$ of Figure 6.

The sequence $H_{10} \rightarrow_{u_1} H'_{10} \ldots H_{13}$ of (b) shows the corresponding processing of these same messages in the evolution from $H_{10}$. Comparing the two sequences, we make two of observations: Firstly that any message, $v$ or $\kappa$, that has been sent from $U$ for receipt by a service agent, $V$ or $W$, will in the (a) sequence be directly received by the service agent in an event $u_9$ or $w_7$; whereas in the (b) sequence the message must first pass through the censor. Thus for an event such as $u_9$ or $w_7$ in (a) there is a corresponding event sequence for (b) comprising two events, $(u_1, u_9)$ or $(u_2, w_7)$. Where the message is a non-terminal message, $v$ in event $u_1$, the censor agent processing it, $X$, creates successor agent $X'$ to process any subsequent message; whereas if the message is a terminal message, $\kappa$ in event $w_2$, there can be no subsequent message and thus no successor censor agent is needed.

Secondly, it will be the case that any system (e.g. an $H_n$ or $H'_n$) arising from $D_0$ in composition with any observer, will be a transformation of a corresponding system ($G_n$) arising from $C_0$ in composition with the same observer. The $H_n$ (or $H'_n$) is produced from $G_n$ by (i) interposing
Figure 7
chains of censor agents on zero or more of the links of \( G_n \), and (ii) applying an appropriate
renaming of ports to make structurally corresponding agents actually identical. For example, \( G_{10} \)
is transformed to \( H_{10} \) by: interposing \( X \) on the link between \( v \) and \( V \), and \( X_1 \) on the link
between \( \kappa \) and \( W \); and renaming those ports by which the various objects in \( G_{10} \) are linked,
and those ports forming \( M' \)'s generator. Although in this case a chain is a single censor agent, it is
possible to construct observer systems which use \( N \) in such a way as to give chains of multiple
censor agents.

The second observation is the crux of the proof for it allows us to define a family of relations
which serve as the bisimulations needed. For some observer \( <W, \Delta> \), the relation \( \Sigma_\Delta \) is the set
of all pairs of systems \( (G,H) \) for which the objects of \( H \) can be derived from those of \( G \) by a
transformation of the nature just described, and the script of \( G (H) \) is the composition of \( \Delta \) and
the behaviour specifications for \( N (M \) and \( X) \).

**Behaviour Specifications**

To characterise \( \Sigma_\Delta \) we first need to formulate the behaviour specifications governing the
behaviours of \( N, M \) and \( X \), for which the behaviour identifiers used will be \( ID_m \) (for both \( M \)
and \( N) \) and \( ID_x \). (Note: The use of the same behaviour identifier \( ID_m \) in both systems simplifies
the proof and is without loss of generality since observational equivalence is preserved by
relabelling of behaviour identifiers.)

For an agent with \( ID_m \), such as \( M \) or \( M' \), the required behaviour specification, \( A_m \), to
prescribe its possible events, such as \( w_8 \), is:

\[
A_m: \quad (ID_m; g; p; q; \quad (r; s; t; ):p \\
(\text{TRUE}) \rightarrow (ID_m; 1; 1)(g; r; (g.1); \quad (g.1); s; q;)
\]

For an agent with \( ID_m \), such as \( N \) or \( N' \), the required behaviour specification, \( A_n \), to
prescribe its possible events, such as \( w_8' \), is:

\[
A_n: \quad (ID_m; g; p; q; \quad (r; s; t; ):p \\
(\text{TRUE}) \rightarrow (ID_m; 1; 2)(g; r; (g.2); ) (ID_x; 2; 2)(g; r; (g.1); ) (-(g.2), (g.1); q;)
\]

The behaviour of an agent such as \( X \) or \( X_1 \) is more complex because it must allow the receipt
and forwarding of a message with any structure (that is with any number of imports, exports and
data items) - in \( G_{10} \) an appropriate \( U \) and \( V \) as observer agents could communicate any
arbitrary message as \( v \), and so the agent \( X \) in \( H_{10} \) must be able to handle any message in an
event such as \( u_1 \). For an agent with \( ID_x \) we define a family of behaviour specifications, \( A_{i;j;k} \),
parameterised by \( i \) for the number of imports in the received message, \( j \) for the number of
exports (\( j>1 \) as there must be a destination output) and \( k \) for the number of data items. Using
\( in, out \) and \( x \) to mean sequences of variables satisfying \( #(in) = i-1, #(out) = j-1, #(x) = k \), we
define:

for \( i>0 \) (that is an event such as \( u_1 \), producing a forwarded message and a successor agent),
\[ A_{ij;k} : \quad (ID_{x;i}, g; p; q) \quad (r, \text{ in}; s, \text{ out}; x): p \]

\[
\text{(TRUE)} \rightarrow \quad (ID_{x;[1:1](g)}; r; (g, 1)); \quad (\langle g, 1 \rangle, \text{ in}; q, \text{ out}; x)
\]

for \( i = 0 \) (that is an event such as \( u_2 \), producing just a forwarded message),

\[ A_{0;j;k} : \quad (ID_{x; g; p}; q) \quad (r, \text{ in}; x): p \]

\[
\text{(TRUE)} \rightarrow \quad (\langle g, 1 \rangle, \text{ in}; q, \text{ out}; x)
\]

We will denote \( A_X = \{A_{ij;k} : i, k \geq 0 \land j \geq 1\} \).

The Transformation Relations

To define the \( \Sigma_\Delta \) relations informally described above we need some auxiliary notions for dealing with chains of censor agents (X-chains).

An \( X\)-chain is any admissible finite non-empty set of agents \( \xi = \{X_1, \ldots, X_k\} \) such that:

(i) \( X_i = (ID_{x;i}, g_i; q_i, p_i) \) for \( i = 1, \ldots, k \);

(ii) \( -q_1 = p_2, \ldots, -q_{k-1} = p_k \) and \( -q_k = p_1 \)

(that is the output of one agent is linked to inport of the previous agent, and the chain is not a closed cycle).

We will denote \( \text{in}(\xi) = q_k \) and \( \text{out}(\xi) = p_1 \).

For some set of \( X\)-chains, \( \Xi \), and some set of objects, \( W \), we will use \( W \uparrow \Xi \) to denote the result of interposing each chain of \( \Xi \) on the appropriate link of \( W \).

Let \( \Xi = \{\xi_1, \ldots, \xi_m\} \) be a non-empty set of \( X\)-chains, and let \( W \) be an admissible set of objects satisfying:

(i) \( \text{out}(\xi_1), \ldots, \text{out}(\xi_m) \) are different outports appearing in outport sequences of objects of \( W \);

(ii) \( \xi_1 \cup \ldots \cup \xi_m \in \text{Adm} \);

(iii) \( P(W) \cap P(\xi_1 \cup \ldots \cup \xi_m) = \{\text{out}(\xi_1), \ldots, \text{out}(\xi_m)\} \);

(iv) \( -\text{in}(\xi_i), \ldots, -\text{in}(\xi_m) \notin P(W) \).

Then we denote by \( W \uparrow \Xi \) the set \( W \uparrow \xi_1 \cup \ldots \cup \xi_m \), where \( W \uparrow \Xi \) is obtained from \( W \) by substituting the outports \( -\text{in}(\xi_i), \ldots, -\text{in}(\xi_m) \) for \( \text{out}(\xi_i), \ldots, \text{out}(\xi_m) \), respectively.

Also, we define \( W \uparrow \emptyset = W \).

Note that the conditions (i)-(iv) imply that \( W \uparrow \Xi \in \text{Adm} \).

For every outport \( \rho \in P(W) \) we will denote by \( \text{length}_\Xi(\rho) \) the length of the chain of censor agents interposed on the \( \rho \) link.

That is, \( \text{length}_\Xi(\text{out}(\xi_i)) = |\xi_i| \), for \( i = 1, \ldots, m \); and \( \text{length}_\Xi(\rho) = 0 \) for \( \rho \not\in \{\text{out}(\xi_1), \ldots, \text{out}(\xi_m)\} \).

Suppose now that \( \Delta \) is a script such that \( ID_m, ID_X \notin ID(\Delta) \). We define \( \Sigma_\Delta \) as the set of all pairs \((G,H)\) of closed systems such that:

(i) \( ID_x \notin ID(G) \);

(ii) \( \Gamma_G = \Delta \cup \{A_m\} \);

(iii) \( \Gamma_H = \Delta \cup \{A_m \cup A_X\} \);
(iv) \( W_H = (W_G \uparrow \Sigma)[h] \) for some set of \( X \)-chains \( \Sigma \) and some renaming of ports \( h \) preserving port generators in all agents of \( W_G \) with behaviour identifiers different from \( ID_m \).

For the purposes of subsequently proving that \( \Sigma_\Delta \) is a bisimulation, we introduce the following lemma.

**Lemma:** Let \((G,H) \in \Sigma_\Delta\) and \( W_H = (W_G \uparrow \Sigma)[h] \).

(i) If \( H \rightarrow_w H' \) and \( w \in \Psi(A_X) \), then \((G,H) \in \Sigma_\Delta\).

(ii) For every message \( \mu \in W_G \) such that \(-\text{dest}(\mu) \in P(\alpha)\) for some agent \( \alpha \in W_G \), there is some \( t \in \Psi(A_X)^* \) such that \( H \rightarrow_t H' \), \((G,H') \in \Sigma_\Delta\) and \( \text{length}_\Sigma(\text{dest}(\mu)) = 0 \), where \( W_{H'} = (W_G \uparrow \Sigma)[h] \).

In proving (i) we have to consider two cases:

**Case 1:** \( w = (((\mu',a),V) \in \Psi((A_0;j,k : j \geq 1 \land k \geq 0)) \) (event \( u_2 \) of Figure 7). Let \( \mu \in W_G \) be such that \( ((\mu_2)\Sigma)[h] = \mu \). Then we have \( W_{H'} = (W_G \uparrow \Sigma)[h] \), where \( \Sigma' \) satisfies:

\[
\text{length}_\Sigma(\text{dest}(\mu)) = \text{length}_\Sigma(\text{dest}(\mu)) - 1; \quad \text{length}_\Sigma(p) = \text{length}_\Sigma(p) \text{ for } p \neq \text{dest}(\mu).
\]

**Case 2:** \( w = (((\mu',a),V) \in \Psi((A_0;j,k : j \geq 1 \land k \geq 0)) \) (event \( u_1 \) of Figure 7). Let \( \mu \) be as above.

Then we have \( W_{H'} = (W_G \uparrow \Sigma)[h] \), where \( \Sigma' \) satisfies:

\[
\text{length}_\Sigma(\text{dest}(\mu)) = \text{length}_\Sigma(\text{dest}(\mu)) - 1; \quad \text{length}_\Sigma(-\cdot) = \text{length}_\Sigma(-\cdot) + 1 \text{ (} q \text{ is the first import in the import sequence of} \mu\text{); and } \text{length}_\Sigma(p) = \text{length}_\Sigma(p) \text{ for } p \neq \text{dest}(\mu), -q.
\]

Note that \(-q = \text{dest}(\mu)\), then \( \text{length}_\Sigma(p) = \text{length}_\Sigma(p) \) for all \( p \).

To prove (ii) we assume \( \text{length}_\Sigma(\text{dest}(\mu)) > 0 \). Then there is \( w = (((\mu',a),V) \in \Psi(A_X) \) such that \( ((\mu_2)\Sigma)[h] = \mu \) and \( H \rightarrow_w H' \). Hence we can repeat the above case analysis obtaining:

\[
W_{H'} = (W_G \uparrow \Sigma)[h] \text{ and } \text{length}_\Sigma(\text{dest}(\mu)) = \text{length}_\Sigma(\text{dest}(\mu)) - 1
\]

(note that \(-\text{dest}(\mu) \in P(\alpha)\) for some agent \( \alpha \in W_G \) excludes the \(-q = \text{dest}(\mu)\) situation in Case 2).

**Proof of the Transformation Relation Being a Bisimulation**

To demonstrate the \( \Sigma_\Delta \) relation is a \( \Delta \)-bisimulation it suffices to show that if \((G,H) \in \Sigma_\Delta\) then:

(A) if \( w \in \Psi(T_G) \) and \( G \rightarrow_w G' \) then there is \( t \in \Psi(T_H)^* \) and a system \( H' \) such that:

\[H \rightarrow_t H', (w) \sim_\Delta t \text{ and } (G',H') \in \Sigma_\Delta;\]

(B) if \( v \in \Psi(T_H) \) and \( H \rightarrow_v H' \) then there is \( s \in \Psi(T_G)^* \) and a system \( G' \) such that:

\[G \rightarrow_s G', (v) \sim_\Delta s \text{ and } (G',H') \in \Sigma_\Delta.
\]

Let \((G,H) \in \Sigma_\Delta\) and let \( W_H = (W_G \uparrow \Sigma)[h] \).

To show (A) we consider three possible classes of events:

A(i) \( w = (U,V) \in \Psi(\Delta) \) is either a spontaneous event or \( U = (\mu,a), \) where \( \text{length}_\Sigma(\text{dest}(\mu)) = 0 \). Then we take \( t = (w_2[h]) \), where \( w_2 = (U_2,V_2) \), and obtain:

\((G',H') \in \Sigma_\Delta \text{ and } W_{H'} = (W_G \uparrow \Sigma)[h]\). We also have \( (w) \sim_\Delta t \text{ since } h \text{ preserves port generators in all agents of } W_G \text{ with behaviour specifications different from } ID_m.\)

A(ii) \( w = ((\mu,a),V) \in \Psi(A_m) \), where \( \text{length}_\Sigma(\text{dest}(\mu)) = 0 \). Let \( \mu = (p,q; r) \). Then we take \( t = (w_3[h]) \), and obtain:

\((G',H') \in \Sigma_\Delta \text{ and } W_{H'} = (W_G \uparrow \Sigma)[h]\), where \( \Sigma' \) is
such that $\text{length}_\Sigma(-q) = 1 + \text{length}_\Sigma(-q)$ and $\text{length}_\Sigma(z) = \text{length}_\Sigma(z)$ for $z \neq -q$. Clearly, $w|_\Delta = () = w_0[h]|_\Delta$, so $(w) \sim_\Delta t$.

A(iii) $w = ((μ, a), V) \in \Psi(T_G)$ and $\text{length}_\Sigma(\text{dest}(μ)) > 0$. Then, by Lemma (ii), there is some $t \in \Psi(A_X)$ such that $H \rightarrow_1 H'$, $(G, H') \in \Sigma_\Delta$ and $\text{length}_\Sigma(\text{dest}(μ)) = 0$, where $W_{H'} = (W_G \upharpoonright \Sigma)[h]$. Note that $t|_\Delta = ()$. Next, if $w \in \Psi(Δ)$ then we proceed as in A(i); and if $w \in \Psi(A_m)$ then we proceed as in A(ii).

To show (B) we consider three possible classes of events:

B(i) $v \in \Psi(A_X)$. Then the thesis follows from Lemma (i) and $v|_\Delta = ()$.

B(ii) $v \in \Psi(Δ)$. Then there is $w \in \Psi(Δ)$ such that $v = w_0[h]$ and $G \rightarrow_\omega G'$, and we can use the same arguments as in A(i).

B(iii) $v \in \Psi(A_m)$. Then there is $w \in \Psi(A_m)$ such that $v = w_0[h]$ and $G \rightarrow_\omega G'$, and we can use the same arguments as in A(ii).

Proof of Observational Equivalence

Having demonstrated that any $\Sigma_\Delta$ is $\Delta$-bisimulation, we can now easily prove that $C_0 = D_0$, where $C_0$ and $D_0$ are as in Figure 6, namely

$$C_0 = \langle (ID_m; (m1_{outport}, m2_{outport}, \ldots); a_{inport}; b_{outport}); (A_m) >$$

and

$$D_0 = \langle (ID_m; (m1_{outport}, m2_{outport}, \ldots); a_{inport}; b_{outport}); (A_n \cup A_X) >$$

For any system $O = (W; Δ)$ such that $G_0 = (C_0 \oplus O)$ and $H_0 = (D_0 \oplus O)$ are defined and give closed systems, there exists a bisimulation, namely $\Sigma_\Delta$, for which $(G_0, H_0) \in \Sigma_\Delta$, since $(W_{G_0} \upharpoonright \emptyset)[h] = W_{H_0}$, where $h(p) = p$ for all $p$.

Extension to Multiple Users and Files

Finally we consider an extension to $C_0$ and $D_0$ such that the M and N agents can be dynamically linked to multiple users and files, as shown (for $M$ only) in Figure 8. We extend $M$ and $N$ agents to allow three kinds of messages each of which has: an integer, $n$, indicating the type of message; and (in addition to the destination port), either an import or an output. The three types of message are:

$n = -1$ means introduce a new user to $M$, with the import allowing the user to communicate to $M$ (e.g. event $v_2$ of Figure 8);

$n = -2$ means introduce a new file to $M$, with the output allowing $M$ to communicate to the file (e.g. event $v_4$ of Figure 8);

$n > 0$ means connect to the $n$-th introduced file, e.g. event $v_6$ of Figure 8 ($n = 1$, which corresponds to the only message that $M$ could receive in the simpler version).

In modifying the above proof for this case, the construction of the family of bisimulations remains unchanged. We would just need to give the new behaviour specifications (see below), modify the
case analyses $A(iii)$ and $B(iii)$ and cover the two new cases for the first and second types of messages for $M$.

![Diagram](image)

**Figure 8**

Let $in = (p_1, \ldots, p_k)$ and $out = (q_1, \ldots, q_l)$ be two sequences of import and output variables, respectively. We will use the notations $in[i\leftarrow z]$ and $out[j\leftarrow y]$, where $i \leq k$, $j \leq l$, $z$ is an import expression and $y$ is an output expression, to denote the following two sequences:

$$in[i\leftarrow z] = (p_1, \ldots, p_{i-1}, z, p_{i+1}, \ldots, p_k)$$

and

$$out[j\leftarrow y] = (q_1, \ldots, q_{j-1}, y, q_{j+1}, \ldots, q_l).$$

The new specifications for agent $M$ are the following; in which $k$ is the number of imports by which $M$ is linked to users, $l$ is the number of outputs by which $M$ is linked to files, $i$ identifies on which particular import a message is received and $j$ (for connection events) identifies the position in $M$’s output sequence the output by which $M$ is linked to the required file.

For $k \geq 1$, $l \geq 0$ and $1 \leq i \leq k$,

**IntrUser$_{k,l,i}$:**

$$(ID_m; g; in; out; ) \ (p; r; q; n); p_i \quad (n = -1) \rightarrow \quad (ID_m; [1:0]; g; in[i-p], r; out; )$$

For $k \geq 1$, $l \geq 0$ and $1 \leq i \leq k$,

**IntrFile$_{k,l,i}$:**
\[(ID_m; g; in; out; ) \quad (p; q; r; n); p_i \quad (n = \cdot 2) \rightarrow \quad (ID_m; [1:0](g); in[i-p]; out, r; )\]

For \(k \geq 1\), \(l \geq 1\), \(1 \leq i \leq k\), and \(1 \leq j \leq l\),

\[M_{\text{Connect}_{k:l;i:j}}:\]
\[(ID_m; g; in; out; ) \quad (p, r; q; n); p_i \quad (n = j) \rightarrow \]
\[(ID_m; [1:1](g); in[i-p]; out[j-(g.1)]; ) \quad (\langle g.1, r; q \rangle)\]

For agent \(N\), there are the same specifications \( IntrUser_{k:l;i} \) and \( IntrFile_{k:l;i} \), but the last specification, \( M_{\text{Connect}_{k:l;i:j}} \), becomes:

\[N_{\text{Connect}_{k:l;i:j}}:\]
\[(ID_m; g; in; out; ) \quad (p, r; q; n); p_i \quad (n = j) \rightarrow \]
\[(ID_m; [1:2](g); in[i-p]; out[j-(g.2)]; ) \quad (IDx; [2:2](g); r; (g.1); ) \quad (\langle g.1, \langle g.2, q \rangle)\]