Journal of Universal Computer Science, vol. 9, no. 1 (2003), 2-33
submitted: 21/2/02, accepted: 24/1/03, appeared: 28/1/03 © J.UCS

A Multiply Hierarchical Automaton Semantics for the IWIM Coordination Model

R. Banach
(Computer Science Dept., Manchester University, Manchester, M13 9PL, U.K.
`banach@cs.man.ac.uk`)

F. Arbab
(Software Engineering Dept., CWI, Kruislaan 413, 1098 SJ Amsterdam, Netherlands
`farhad@cwi.nl`)

G. A. Papadopoulos
(Computer Science Dept., University of Cyprus, 75 Kallipoleos St., Nicosia, Cyprus
`george@cs.ucy.ac.cy`)

J. R. W. Glauert
(School of Information Systems, University of East Anglia, Norwich, NR4 7TJ, U.K.
`J.Glauert@sys.uea.ac.uk`)

Abstract: The drawbacks of programming coordination activities directly within the applications software that needs them are briefly reviewed. Coordination programming helps to separate concerns, making complex coordination protocols into standalone entities; permitting separate development, verification, maintenance, and reuse. The IWIM coordination model is described, and a formal automata theoretic version of the model is developed, capturing the essentials of the framework in a fibration based approach. Specifically, families of worker automata have their communication governed by a state of a manager automaton, whose transitions correspond to reconfigurations. To capture the generality of processes in IWIM systems, the construction is generalised so that process automata can display both manager and worker traits. The relationship with other formalisations of the IWIM conception of the coordination principle is explored.
Keywords: Coordination, IWIM, Automata, Fibration.
Categories: C.2.4, D.1.3, D.2.6, D.3.3, F.1.1.

1 Introduction

The massively parallel systems that can be built today require programming models that explicitly deal with the concurrency of cooperation among large numbers of entities in a single application. Today's concurrent applications typically use ad hoc templates to coordinate the cooperation of their components, and this is symptomatic of a lack of proper coordination frameworks for describing complex cooperation protocols in terms of simple primitives and structuring constructs.

In most real applications, there is no paradigm in which we can systematically talk about cooperation of active entities, and in which we can compose cooperation scenarios such as client-server, workers pool, etc., out of a set of more basic concepts. Consequently, applications programmers must deal directly with the lower-level communication primitives that instantiate the cooperation model of a concurrent application. These primitives are generally scattered throughout the source code, interspersed with non-communication application code, and the cooperation model never manifests itself in a tangible form. Thus it is not an identifiable piece of source code that can be designed, developed, debugged, maintained, and reused, in isolation from the rest of the application. This inability to deal with the cooperation model of a concurrent application explicitly, contributes to the difficulty of developing working concurrent applications containing large numbers of actively cooperating entities.

Despite the fact that the implementation of complex protocols is often the most difficult part of a development, the end result is typically so nebulous that it cannot be recognized as a commodity in its own right. This makes maintenance and modification of the cooperation protocols much more difficult than necessary, and their reuse next to impossible.

The two most popular models of communication within highly concurrent applications are shared memory and message passing. In the shared memory model, interprocess synchronisation primitives play the dominant role, with interprocess communication subordinate, whereas in the message passing model, interprocess communication is dominant, and synchronisation subordinate. The latter makes the message passing model somewhat more flexible than the shared memory model and, therefore, it is the dominant model used in concurrent applications. However, both paradigms are too low-level to serve as a proper foundation for systematic construction of cooperation protocols as explicit, tangible pieces of software.

Such observations have led in recent years to an upsurge in activity in so-called coordination frameworks and languages. An early survey is [Malone and Crowston (1994)] which characterisies coordination as an emerging discipline. Various approaches with roots in eg. the actor model [Agha (1986)], or in logic programming [Shapiro (1989)], were instrumental in establishing coordination as an independent discipline. See [Ciancarini and Hankin (1996), Garlan and Le Metayer (1997), Papadopoulos and Arbab (1998), Ciancarini and Wolf (1999), Porto and Roman (2000), Omicini et al. (2002)] for representative contemporary work. A number of higher level perspectives have emerged. Among these are the tuple based approaches such as Linda [Gelernter (1985), Carriero and Gelernter (1989)], and by contrast, the connection control based approaches amongst which we find the IWIM (Ideal Worker Ideal Manager) model. It is with this model that this paper is concerned.

The rest of this paper contains the following. In [Section 2] we survey the IWIM model informally. With this motivation covered, in [Section 3] we develop a theoretical automaton-based model for IWIM, which we call the IWIM systems model. This is developed gradually, as it is a fairly complicated construction, aiming to reflect the essentials of IWIM in a credible manner. The underlying idea is that families of worker automata perform their tasks under the supervision of a manager automaton. Change of state of the manager corresponds to reconfiguration, whereupon a different family of worker automata shoulders the burden. This basic idea is elaborated to enable arbitrarily complex hierarchies to be modelled. Although our model is reasonably involved, it falls short of trying to capture everything about IWIM or any specific implementation of the IWIM idea, such as is to be found in the formal specification of the MANIFOLD language [Arbab et al. (1993), Bonsangue et al. (2000)]. In particular we abstract away from the ability of workers to continue with internal actions on their own, which in the full IWIM model they can do irrespective of the attentions of any manager. One principal purpose of this work could be seen as exploring the viability of fibration based ideas in the arena of reconfiguration problems.

In [Section 4] we discuss how the instantaneous reconfiguration aspect of our IWIM systems can be generalised to model the asynchronous event based reconfigurations characteristic of real IWIM frameworks. In [Section 5] we show how the model of Arbab, de Boer and Bonsangue [Arbab et al. (2000a)], a model featuring aspects of reconfiguration, can be expressed by IWIM systems; and in [Section 6] we show how the model of Katis, Sabadini and Walters [Katis et al. (2000)], a significantly different theoretical account, can also be captured within IWIM systems. These two enterprises support the other principal purpose of this work, which is to explore how the IWIM idea may be formalised in a manner that vividly highlights the special nature of the relationship between managers and workers in IWIM, and to compare such a formalization with models that do not do so. One aspect of IWIM systems not covered in this paper is the issue of their algebraic properties. The highly structured IWIM systems model has a rich algebraic theory. However an in depth account would almost double the size of this paper; see [Banach et al. (2002)]. [Section 7] concludes.

2 The IWIM Model

In this section we review the generic coordination framework known as the Ideal Worker Ideal Manager (IWIM) model [Arbab (1995), Arbab (1996), Arbab et al. (1998)]. The basic concepts in the IWIM model are processes, events, ports, and channels. A process is a black box with well defined ports of connection through which it exchanges units of information with the other processes in its environment. A port is a named opening in the bounding walls of a process through which units of information are exchanged using standard I/O primitives such as read and write; we assume that each port is used for the exchange of information in only one direction: either into the process (input port) or out of the process (output port).

The interconnections between the ports of processes are made through channels. A channel connects a port of a producer process to a port of a consumer process. Independent of the channels, there is an event mechanism for information exchange in IWIM. Events are broadcast by their sources into their environment, yielding event occurrences. In principle, any process in an environment can pick up a broadcast event occurrence. In practice, usually only a few processes pick up occurrences of each event, because only they are tuned in to the relevant sources.

The IWIM model supports anonymous communication: in general, a process does not, and need not, know the identity of the processes with which it exchanges information. This concept reduces the dependence of a process on its environment and makes processes more reusable; it also makes the protocols governing such communication more reusable.

A process in IWIM can be regarded as a worker process or a manager (or coordinator) process. The responsibility of a worker process is to perform a task. A worker process is not responsible for the communication that is necessary for it to obtain the proper input it requires to perform its task, nor is it responsible for the communication that is necessary to deliver the results it produces to their proper recipients. In general, no process in IWIM is responsible for its own communication with other processes. It is always the responsibility of a manager process to arrange for and to coordinate the necessary communications among a set of worker processes.

There is always a bottom layer of worker processes, called atomic workers, in an application. In the IWIM model, an application is built as a (dynamic) hierarchy of worker and manager processes on top of this layer. Aside from the atomic workers, the categorization of a process as a worker or a manager process is subjective: a manager process *proc* that coordinates the communication among a number of worker processes, may itself be considered as a worker process by another manager process responsible for coordinating the communication of *proc* with other processes.

In IWIM, a channel is a communication link that carries a sequence of bits, grouped into units. A channel represents a reliable, directed, and perhaps buffered, flow of information in time. Here, reliable means that the bits placed into a channel are guaranteed to flow through without loss, error, or duplication, and with their order preserved; and directed means that there are always two identifiable ends in a channel: a source and a sink. Once a channel is established between a producer process and a consumer process, it operates autonomously and transfers the units from its source to its sink.

If we make no assumptions about the internal operation of the producer and the consumer of a channel *c*, we must consider the possibility that *c* may contain some pending units. The pending units of a channel *c* are the units that have already been delivered to *c* by its producer, but not yet delivered by *c* to its consumer. The possibility of the existence of pending units in a channel gives it an identity of its own, independent of its producer and consumer. It makes it meaningful for a channel to remain connected at one of its ends, after it is disconnected from the other. The full details of the IWIM model codify a number of variations on this theme, but for our purposes, a channel will stay alive as long as one end or another is connected to a process.

Worker processes have two means of communication: via ports, and via events. The communication primitives that allow a process to exchange data through its ports are conventional read and write primitives. A process can attempt to read data from one of its input ports. It hangs if no data is presently available through that port, and continues once data is made available. Similarly, a process can attempt to write data to one of its output ports. It hangs if the port is presently not connected to any channel, and continues once a channel connection is made to accept the data.

It is worth mentioning at this point that the interaction of all the ideas sketched in the preceding paragraphs conspires to make the notion of port quite intricate, as the formal models of subsequent sections show. The fact that an individual port *p* belongs to a specific worker (which may also be engaged in management activities, but none involving *p*), but has its connectivity controlled by a different process, whose interest in *p* may wax and wane depending on the state of the computation, requires careful modelling to ensure that there are ‘no bits left dangling’. Thus various aspects of a port’s functionality end up attached to different parts of the formal model, the whole being subject to a number of carefully constructed invariants.

Besides reading and writing over ports, a process *proc* can also broadcast an event *e* to all other processes in its environment by raising that event. The identity of the event *e* together with the identity of the process *proc* comprise the event occurrence. A process can also pick up event occurrences broadcast by other processes and react to them.

Certain events are guaranteed to be broadcast in special circumstances; for example, termination of a process instance always raises a special event to indicate its death. Our formal model in the rest of the paper will be quite limited in that we only model reconfiguration events. Even then, for simplicity, the modelling will be synchronous, a defect we address later.

A manager process can create new instances of processes (including itself) and broadcast and react to event occurrences. It can also create and destroy channel connections between various ports of the process instances it knows, including its own. Creation of new process instances, as well as installation and dismantling of communication channels are done dynamically. Specifically, these actions may be prompted by event occurrences it detects. Each manager process typically controls the communications among a dynamic family of process instances in a data-flow like network. The processes themselves are generally unaware of their patterns of communication, which may change in time, according to the decisions of a coordinator process.

In our formal model, again for reasons of simplicity, we eschew the full generality of these concepts. Our process networks will turn out to be statically defined, though the execution trajectory through this stucture will be dynamically determined. As such they may be viewed as the static unwinding of an implicit but more succinct syntactic specification of dynamic behaviour, and the unwinding enables us to restrict discussion to the semantic level alone, a welcome simplification.

3 IWIM Automata

In this section, we distil the essentials of the ideas just described, to create the model which will serve as the basis for the semantics of IWIM in the rest of the paper. We build the model up in two steps. The first is based on a fibration-inspired strategy, to reflect the way that IWIM events tear down and rebuild interconnections between families of processes. Accordingly, elementary IWIM automata will have in the base a manager automaton, describing how the manager part of an elementary IWIM system moves, and above each state of the manager automaton, there will be a collection of worker automata, connected together according to the prescription contained in the manager state. The various worker collections are then integrated into a single elementary IWIM system using an 'above' relation describing how workers relate to states of the manager, a construction inspired in essence by the Grothendieck construction. As a result of this, each configuration of the overall automaton can be projected down onto the relevant state of the manager in the manner of a fibration[1].

The capacity of IWIM systems to reconfigure themselves via events that provoke managers into reconfiguration activities, is here modelled by mappings of certain worker moves (that represent the raising of the event) to manager moves (that represent the reception and processing of the event, resulting in reconfiguration). Unlike genuine IWIM systems, this is a synchronous activity in our model, but we will show in Section 4 that the asynchronous aspects can be recaptured within our framework.

[Fig. 1] illustrates in pictures what we have just described in words for elementary IWIM automata. It shows a collection of worker automata $\{A, B, C, D, E, S\}$ sitting

1. The projection oriented nature of fibrations explains why we say 'above' and not 'below', cf. Proposition 3.4 and Proposition 3.8 below.

above a manager *Man*, forming an elementary IWIM system. The states of *Man* i.e. {*l*, *m*, *n*}, each map to communication networks consisting of directed graphs of ports and channels. The ports of these networks correspond bijectively to input and output ports in the workers, who are ignorant of whence come their input messages and where their output messages are destined. Input ports are shown solid, while output ports are hollow. Furthermore these bijections in large part mimic the substructuring of individual ports in IWIM into their private and public parts. Also, following these bijections up to the workers reveals which workers are above which management states. Note that worker *B* is above more than one management state. This means that when *Man* makes a transition from *l* to *m*, *B* is unaffected and continues to work as before. Attached to each channel is a queue of messages, illustrated for just one channel for *l* in the figure. Some of the channels can be external, such as the external input channel for state *l*, and

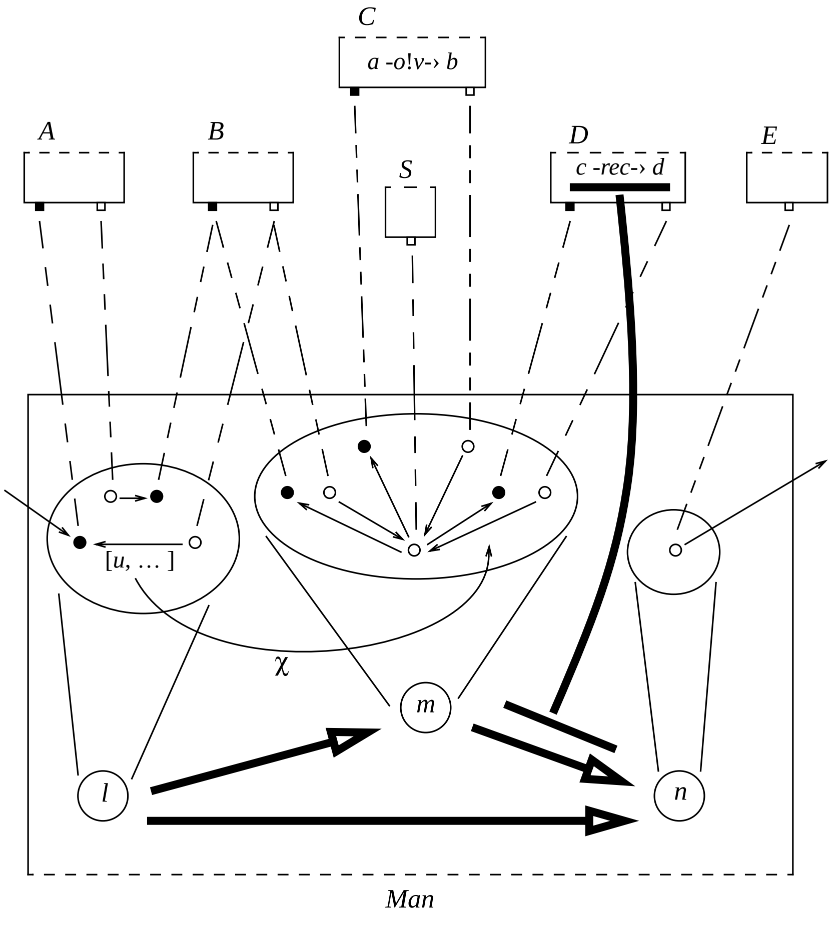

Fig. 1: *A manager Man with some workers above it. Broken vertical lines show the correspondence between worker and manager ports, bijective for each manager state.*

the external output channel for n; these allow connection to and exchange of information with the outside world. Note however that external input can only take place when l is the current management state, and external output can only take place when n is the current management state. The management transitions must specify what happens to the message queues. These are mapped by additional data illustrated by χ in the figure and merged into the destination queues.

Worker C shows a typical worker output transition; there are similar worker input transitions. The port of worker S shows that ports are really quite general purpose concepts in IWIM, able to accomodate several incoming and outgoing channels. Worker S itself can be seen as providing a serialisation service for B, C, D. Worker D shows a reconfiguration event transition. The thick line from the transition to the manager illustrates that the atomic transition label *rec* is mapped to the manager transition from m to n. In this manner the workers can provoke reconfigurations implemented by the manager.

In the second step of the two step strategy for building our IWIM system model, the elementary IWIM system construction just described is generalised to take account of the more flexible nature of real IWIM systems. Now, processes may manifest both manager and worker roles, worker processes may enjoy the attentions of more than one manager, and manager processes may enjoy the benefits of more than one worker. To cope with this, we define IWIM worker-manager automata as asynchronous products of individual worker and manager automata. Also the relation connecting workers and managers becomes global. In this manner we get unrestricted IWIM systems. The previously mentioned properties continue to hold. In particular, configurations of an unrestricted IWIM system can be projected down onto configurations of their mangers.

Let us illustrate all this in another figure. [Fig 2] shows four worker-manager automata, W, X, Y, Z. These are drawn as rectangles with the dashed horizontal line representing the division between the worker and manager facets, the manager facet being uppermost. The worker structure is suppressed in all cases, and the fact that the manager parts of W and X are empty is intended to indicate that these automata are atomic workers, with trivial manager facets. The arrows emanating from manager states point to the worker facets under their control. [Fig 2] illustrates that (almost) completely general management relationships are permitted between worker-manager automata. In fact the only restriction is that an automaton's manager facet cannot manage it's own worker facet. Of course in realistic settings, the kind of contorted and cyclic dependencies occurring in [Fig. 2] do not really arise. Far more plausible, are regularly structured hierarchies with atomic workers in the bottommost layer.

3.1 Elementary IWIM Systems

Definition 3.1 An IWIM manager automaton is a triple (M, m_{I}, R), where M is a set of management states, $m_{\mathrm{I}} \in M$ is an initial state, and R is a set of reconfiguration transitions. These components are further stuctured as follows. Each management state m is itself the name of a pair (P_m, C_m), where P_m is a set of port names, and C_m is a set of channel names. There are two partial functions $s_m, t_m : C_m \to P_m$ which map channels to source and target port names, whenever s_m or t_m are defined. They satisfy $\mathrm{dom}(s_m) \cup \mathrm{dom}(t_m) = C_m$, i.e. each channel is connected to at least one port — channels not in dom(s_m) are called external input channels, and channels not in dom(t_m) are called external output channels; channels in both dom(s_m) and dom(t_m) are called internal channels. In a reconfiguration transition, written *m -r-› n*, the *r* is shorthand for a partial injection on the channel names $\chi_{m,n} : C_m \rightarrow C_n$. Also for each management state *m*, we have an identity transition *m* -id$_m$-› *m* in which the $\chi_{m,m}$ partial injection is a total identity.

The above definition characterises states of the manager automaton as connection networks in which the ports do not have a unique orientation (as input or output ports). Different states *m*, *n* may refer to the same connection network. Reconfigurations identify some channels of the source state with some channels of the target.

Definition 3.2 An IWIM worker automaton is a triple (*I*, *O*, *A*), where *I* is a set of input ports, *O* is a set of output ports, and *I* and *O* are disjoint. *A* = (*St*, *Init*, *Tr*) is an automaton with states *St*, of which $Init \in St$ is an initial state, and $Tr \subseteq St \times Act \times St$ is a transition relation, where *Act* is a set of actions of the form *in*?*v* or *out*!*v* or *rec*. In the first two kinds of action, $in \in I$, $out \in O$, and we assume that there is a global alphabet of values *Val* containing *v*. In the last kind, *rec* is just a name (intended to be the name of a reconfiguration transition as in Definition 3.1). Where convenient below, we will write transitions using the notation *a -in*?*v-› b* or *a -out*!*v-› b* or *a -rec-› b.* We define

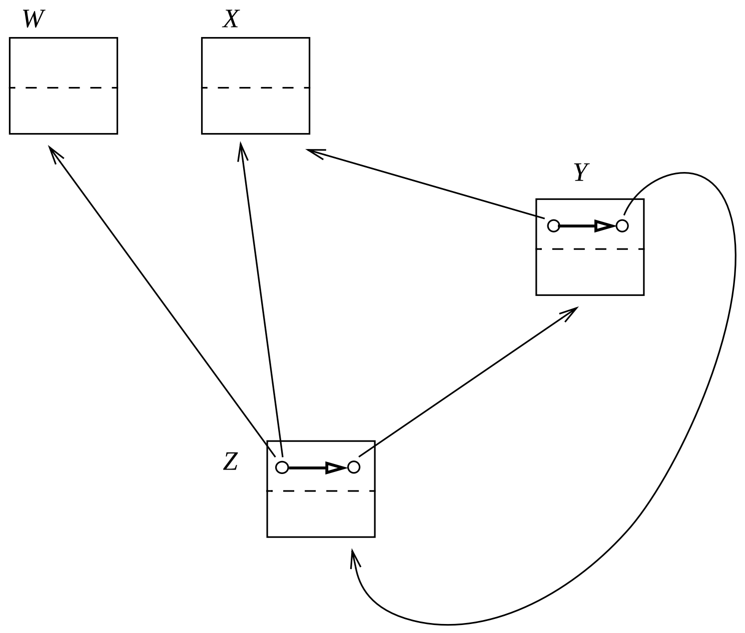

Fig. 2: *A schematic illustration of a network of worker-manager automata. Arrows from manager states to other automata show the 'above' relation of the system. N.B. The fact that Y manages Z and Z also manages Y is legitimate in the IWIM model.*

$Tr_I = \{a \text{ -}in?v\text{-›}\ b \in Tr\}$, $Tr_O = \{a \text{ -}out!v\text{-›}\ b \in Tr\}$, $Tr_R = \{a \text{ -}rec\text{-›}\ b \in Tr\}$, so that $Tr = Tr_I \cup Tr_O \cup Tr_R$, the union being evidently disjoint. Additionally we define $Rec = \{rec \mid a \text{ -}rec\text{-›}\ b \in Tr\}$ the alphabet of reconfiguration events of the worker.

So far, workers are automata of a fairly standard kind. Now we show how workers and managers are glued together.

Definition 3.3 An elementary IWIM system (*Man*, *Wor*) consists of an IWIM manager automaton *Man*, an elementary workforce *Wor*, and ancillary data to be described below. *Wor* is a set of worker names together with a map *wor*, which yields for each worker $w \in Wor$, an IWIM worker automaton $wor(w)$. Furthermore we have:

(1) There is a relation ^ between *Wor* and the management states of *Man*. We write w^m to say that a worker w is *above* a management state m if the pair is in the relation.

(2) If a worker w is above a management state m, then there is a map $r_{w\text{^}m}$ from the *rec* actions of $wor(w)$, into reconfiguration transitions $m \text{ -}r\text{-›}\ n$ of *Man*.

(3) For each management state $m \in Man$, there is a total bijection $\lambda_m : P_m \rightarrow IO_m$ where IO_m is the disjoint union of all of the input and output ports of all workers above m; i.e. $IO_m = \biguplus_{k\text{^}m}\{i \mid i \in I_{wor(k)}\} \uplus \biguplus_{k\text{^}m}\{o \mid o \in O_{wor(k)}\}$.

(4) Associated to each channel $c \in C_m$ (where m is a management state), there is a queue of messages which we write c:$[u_0, u_1, \dots]$. Each u_i is in *Val*. The front of this queue is u_0.

A configuration of an elementary IWIM system (*Man*, *Wor*) consists of:

(1) a state m of *Man*;

(2) a set $ests = \{a_k \mid a_k \in St_{wor(k)}, k \in Wor\}$ of states a_k one for each worker k;

(3) a set $qs = \{c{:}q_c \mid c{:}q_c = c{:}[u_0, u_1, \dots], c \in C_n, n \in M\}$ of queues of messages c:$[u_0, u_1, \dots]$ one for each channel of each management state.

Note that in the above, *ests* may equivalently be viewed as the range of a function which maps each worker to one of its states, so that a_k is formally an ordered pair. Since we are overwhelmingly concerned with the states and how they change, we will not use the more cumbersome functional apparatus. Similar remarks apply to *qs* though here some of the indexing information is routinely suppressed.

A configuration of an elementary IWIM system (*Man*, *Wor*) is initial iff: m is initial, the a_k are also all initial, and the queues associated with all channels are empty.

A transition of an elementary IWIM system (*Man*, *Wor*) in state (m, *ests*, *qs*) is one of the following six kinds:

(ENVI) The environment adds a value to the input end of a queue whose source end is not attached to any port (an external input channel’s queue).

$$\frac{c \notin \mathrm{dom}(s_m)\ ,\quad c \in \mathrm{dom}(t_m)\ ,\quad qs_{\mathrm{rest}} = qs - \{c{:}[\ \ldots\ , u_n]\}}{m \longrightarrow m\ ,\quad ests \longrightarrow ests\ ,\quad qs \longrightarrow qs_{\mathrm{rest}} \cup \{c{:}[\ \ldots\ , u_n\ , u]\}}$$

(ENVO) The environment removes a value from the output end of a queue whose target end is not attached to any port (an external output channel's queue).

$$\frac{c \notin \mathrm{dom}(t_m)\ ,\quad c \in \mathrm{dom}(s_m)\ ,\quad qs_{\mathrm{rest}} = qs - \{c{:}[u, u_1, \ldots\ ]\}}{m \longrightarrow m\ ,\quad ests \longrightarrow ests\ ,\quad qs \longrightarrow qs_{\mathrm{rest}} \cup \{c{:}[u_1, \ldots\ ]\}}$$

(IN) A worker automaton performs an input on one of its input ports, removing the front element from an input queue attached to the port, of which there must be at least one.

$$\frac{k\wedge m\ ,\ a_k \in ests\ ,\ a_k \xrightarrow{i?u} b_k\ ,\quad \lambda_m(p) = i \in I_{wor(k)}\ ,\ t_m(c) = p\ ,\quad ests_{\mathrm{rest}} = ests - \{a_k\}\ ,\quad qs_{\mathrm{rest}} = qs - \{c{:}[u, u_1, \ldots\ ]\}}{m \longrightarrow m\ ,\quad ests \longrightarrow ests_{\mathrm{rest}} \cup \{b_k\}\ ,\quad qs \longrightarrow qs_{\mathrm{rest}} \cup \{c{:}[u_1, \ldots\ ]\}}$$

(OUT) A worker automaton performs an output on one of its output ports, adding a value to the end of any output queue attached to the port, of which there must be at least one.

$$\frac{k\wedge m\ ,\ a_k \in ests\ ,\ a_k \xrightarrow{o!u} b_k\ ,\quad \lambda_m(p) = o \in O_{wor(k)}\ ,\quad \varnothing \neq Out = \{d \mid s_m(d) = p\}\ ,\quad ests_{\mathrm{rest}} = ests - \{a_k\}\ ,\quad qs_{\mathrm{rest}} = qs - \{d{:}[\ \ldots\ , u_{d,n_d}] \mid d \in Out\}}{m \longrightarrow m\ ,\quad ests \longrightarrow ests_{\mathrm{rest}} \cup \{b_k\}\ ,\quad qs \longrightarrow qs_{\mathrm{rest}} \cup \{d{:}[\ \ldots\ , u_{d,n_d}, u] \mid d \in Out\}}$$

(FOR) A port performs a forwarding action, removing the front element from an input queue attached to the port and inserting (a copy of) it to all output queues attached to the port, of which there must be at least one.

$$\frac{\begin{array}{l} t_m(c) = p\ , \\ \varnothing \neq Out = \{d \mid s_m(d) = p\}\ , \\ qs_{\text{rest}} = qs - (\{c{:}[u, u_1, \ldots\ ]\} \cup \{d{:}[\ \ldots, u_{d,n_d}] \mid d \in Out\}) \end{array}}{\begin{array}{l} m \longrightarrow m\ , \\ ests \longrightarrow ests\ , \\ qs \longrightarrow qs_{\text{rest}} \cup \{c{:}[u_1, \ldots\ ]\} \cup \{d{:}[\ \ldots, u_{d,n_d}, u] \mid d \in Out\} \end{array}}$$

NB. The above notation is intended to include the case that $c \in Out$, whereupon the front message of c's queue is moved to its tail.

(REC) A worker automaton k_r performs a *rec* action a_{k_r} *-rec->* b_{k_r}, provoking a reconfiguration m *-r->* n of the elementary IWIM system, given by the function $r_{k_r \wedge m}$. The manager automaton makes a transition to the new state. Worker automaton k_r completes its transition. Worker automata other than k_r who are above both the old and new manager state remain as before. Worker automata above the old but not the new manager state go into suspension. Worker automata not above the old but above the new manager state are awakened. The queues of channels above the old manager state which are reassigned via the channel reconfiguration data are moved according to that data, being merged with the existing queues at target channels and leaving the queues at originating channels empty. The queues at other channels remain as before.

$$\frac{\begin{array}{l} k_r \wedge m\ ,\ a_{k_r} \in ests\ ,\ a_{k_r}\ \text{-}rec\text{->}\ b_{k_r}\ , \\ r_{k_r \wedge m}(rec) = m\ \text{-}r\text{->}\ n = \chi_{m,n} : C_m \to C_n\ , \\ ests_{\text{rest}} = ests - \{a_{k_r}\}\ , \\ qs_{\text{del}} = \{c{:}q_c \mid c \in C_m, c \in \text{dom}(\chi_{m,n})\} \cup \{d{:}q_d \mid d \in C_n, d \in \text{rng}(\chi_{m,n})\}\ , \\ qs_{\text{rest}} = qs - qs_{\text{del}}\ , \\ qs_{\text{dom}} = \{c{:}[\,] \mid c \in C_m, c \in \text{dom}(\chi_{m,n})\}\ , \\ qs_{\text{merge}} = \{d{:}q_{cd} \mid c{:}q_c, c \in C_m, c \in \text{dom}(\chi_{m,n}), \\ \qquad\qquad d{:}q_d, \chi_{m,n}(c) = d \in C_n, d \in \text{rng}(\chi_{m,n}), \\ \qquad\qquad q_{cd} \in \text{merge}(q_c, q_d)\} \end{array}}{\begin{array}{l} m \longrightarrow n\ , \\ ests \longrightarrow ests_{\text{rest}} \cup \{b_{k_r}\}\ , \\ qs \longrightarrow qs_{\text{rest}} \cup qs_{\text{dom}} \cup qs_{\text{merge}} \end{array}}$$

This transition system has some features that deserve comment. Note firstly that input/output and forwarding activities are completely decoupled. For this reason it makes little sense for the manager to connect up a port to use simultaneously as a broadcasting device, and as an input device to the relevant worker, since the input messages and forwarded messages are necessarily disjoint. Thus since even forwarding ports have to belong to some worker, it is best to invent special purpose dummy workers just for the purpose, such as worker S in [Fig. 1].

A second issue concerns the creation and destruction of processes. IWIM is entirely virtuous regarding matters of life and death: there is no murder, only suicide. The most that managers can accomplish is anasthesia. When a reconfiguration transition takes a worker out of the current configuration because that worker is not above the new current management state, the worker sleeps, because being above the current management state is a hypothesis of all six transition types. When the current management state once more becomes one which the worker is above, it wakes and is able to participate in worker transitions again. It is the worker's own responsibility to enter a state out of which no transitions emerge if it wishes to die.

Thirdly there arises the issue of queue management during reconfiguration transitions. We have elected to merge assigned queues with existing ones (for given source and target ports) as representing an abstraction of the potential presence of several independent queues from the source to the target. The latter would require a more complex notion of reconfiguration transition than we wish to get embroiled in.

Let *EConfs*(*Man*, *Wor*) be the set of all configurations of (*Man*, *Wor*). Equipping it with the transitions just described makes it into a transition system. We regard this transition system as unlabelled, it being the case that the kind of step involved is always deducible from the pair of configurations in question.

A run of (*Man*, *Wor*) is, in the normal manner, a sequence of contiguous transitions of *EConfs*(*Man, Wor*), starting with an initial configuration:

$$(m, ests, qs) \longrightarrow (m', ests', qs') \longrightarrow (m'', ests'', qs'') \longrightarrow \ldots$$

Let *Mngr*(*Man*, *Wor*) be the set of manager states of configurations in *EConfs*(*Man*, *Wor*). These are given by a function $e\pi_{man}$ where $e\pi_{man}(m, ests, qs) = m$. The set *Mngr*(*Man*, *Wor*) can be equipped with transitions derived from the (REC) transitions of *EConfs*(*Man*, *Wor*). Thus to the transition $(m, ests, qs) \longrightarrow (m', ests', qs')$ corresponds the *Mngr*(*Man*, *Wor*) transition $e\pi_{man}(m, ests, qs) \longrightarrow e\pi_{man}(m', ests', qs')$, i.e. $m \longrightarrow m'$, (we regard these transition as unlabelled too). We also add an identity transition $m \longrightarrow m$ to each manager state in *Mngr*(*Man*, *Wor*).

Now although a particular worker may be above several manager states, making problematic the definition of a projection from the static structure of the elementary IWIM system to its manager, the same is not true of the set of configurations of the elementary IWIM system and its transition system, *EConfs*(*Man*, *Wor*), as it relates to the set of manager states. In *EConfs*(*Man*, *Wor*), some specific manager state always indexes any worker state that forms part of a configuration, and so we obtain the following result.

Proposition 3.4 Let (*Man*, *Wor*) be an elementary IWIM system. Let *EConfs*(*Man*, *Wor*) be the associated transition system and *Mngr*(*Man, Wor*) be the corresponding set of manager transitions. Then there is a projection:

$$\Pi_e : EConfs(Man, Wor) \to Mngr(Man, Wor)$$

which maps states by:

$$(m, ests, qs) \mapsto m = e\pi_{man}(m, ests, qs)$$

and which maps (REC) transitions by:

$$(m, ests, qs) \longrightarrow (m', ests', qs')$$
$$\mapsto$$
$$m \longrightarrow m' = e\pi_{man}(m, ests, qs) \longrightarrow e\pi_{man}(m', ests', qs')$$

and which maps (ENVI), (ENVO), (IN), (OUT), transitions to identity transitions:

$$\begin{array}{l}(m, ests, qs) \longrightarrow (m, ests', qs') \\ \quad\ \mapsto \\ m \longrightarrow m\end{array}$$

Proof. Obvious. ☺

3.2 Unrestricted IWIM Systems

The previous section captures the essence of the process by which an individual manager automaton manages a group of worker automata. However the IWIM model does not restrict worker management to a single layer. Managers may themselves be workers managed by others, in time honoured hierarchical fashion. We model this here by allowing managers to themselves acquire a worker facet. The result is effectively a product of the two preceding constructions.

Definition 3.5 An IWIM worker-manager automaton is the asynchronous product of an IWIM worker automaton (I, O, A) as in Definition 3.2, and an IWIM manager automaton (M, m_I, R) as in Definition 3.1. That is to say, an IWIM worker-manager automaton is of the form $(I, O, A)\otimes(M, m_I, R)$, where (I, O, A) is called the worker facet and (M, m_I, R) is called the manger facet. The set of states of the worker-manager automaton is $St \times M$, with initial state $(Init, m_I)$, and there are two kinds of transitions: worker transitions, for example (a, m) -*w*-› (b, m), where a -*w*-› b is a transition of (I, O, A) (and the manager facet m remains unchanged), and manager transitions, for example (a, m) -*r*-› (a, n), where m -*r*-› n is a transition of (M, m_I, R) (and the worker facet a remains unchanged).

The following is evident.

Proposition 3.6 An IWIM worker-manager automaton for which the worker facet is a single (initial) state IWIM worker automaton with empty transition relation is strongly bisimilar to an IWIM manager automaton. Also an IWIM worker-manager automaton for which the manager facet is a single (initial) state IWIM manager automaton whose port and channel sets are empty, and with transition relation consisting of just the obligatory (in this case empty) identity function, is strongly bisimilar to an IWIM worker automaton.

In view of this, we can refer to IWIM worker-manager automata with trivial worker facets as pure mangers, and to IWIM worker-manager automata with trivial manager facets as pure workers.

Now that individual automata are capable of both worker and manager behaviour, we can define an unrestricted IWIM system as a community of automata where the manager facets of individual automata manage their individual workforces drawn from the same community, and the worker facets of individual automata each do their jobs coordinated by one or more manager facets, since we place no restriction on the number of bosses any poor labourer might have. In keeping with the best industrial practice, no worker is ever his own manager (no selfdetermination — no one sets their own salary, nor signs off their own expense claims). Since the moves of the whole system are the moves of the individual elements, we need no additional restrictions beyond the no selfdetermination rule and the restrictions that apply to elementary IWIM systems, to have consistency.

Definition 3.7 An unrestricted IWIM system WM is a set of IWIM worker-manager automaton names called WM, a subset $Initial_{WM} \subseteq WM$, together with ancillary data described below. There are three maps: $worman$, wor, man, where for each $wm \in WM$, $worman(wm)$ is an IWIM worker-manager automaton, $wor(wm)$ is its worker facet, and $man(wm)$ is its manager facet. We write m_{wm} to say that state m is a state of a facet of automaton wm, the facet intended being clear from the context; formally m_{wm} is an ordered pair, just as before. The states of a worker-manager automaton wm are thus written (a_{wm}, m_{wm}), where a is the state of the worker facet and m is the state of the manager facet.

Moreover, other aspects of the notation for elementary IWIM systems acquire additional subscripting to indicate what part of the unrestricted IWIM system they refer to. Thus we have $P_{m_{wm}}$ for the set of port names of state m of the manager facet $man(wm)$ of wm; likewise $C_{m_{wm}}$ is the corresponding set of channel names.

There is a binary above relation $\wedge$ where $wm'{\wedge}m_{wm}$ means that the worker facet $wor(wm')$ of automaton wm' is above state m of the nontrivial manger facet $man(wm)$ of automaton wm. The no selfdetermination rule implies that whenever $wm'{\wedge}m_{wm}$, then $wm' \neq wm$. The workforce $\{wm_1, \ldots , wm_n\}$ of automata whose worker facets are above states of the manager facet of wm is refered to as an elementary IWIM subsystem of WM, and is an elementary IWIM system in the sense of Definition 3.3 when we disregard the manger facets of the workers and the worker facet of the manager. Thus $IO_{m_{wm}}$ is the set of input and output ports of the workforce above m_{wm}. Specifically for an elementary IWIM subsystem:

(1) The above relation is inherited from the global one, and we will assume henceforth that no automaton is above the unique state of a trivial manager.

(2) There is a map $r_{wm'{\wedge}m_{wm}}$ of the rec transitions of worker facets into reconfiguration transitions of the corresponding nontrivial manager facet.

(3) The total bijection property of manager ports to workforce input/output ports holds via a map $\lambda_{m_{wm}} : P_{m_{wm}} \rightarrow IO_{m_{wm}}$.

(Note that the no selfdetermination rule is consistent with the asynchronous product structure of the transitions for worker-manager automata. Otherwise some $r_{wm{\wedge}m_{wm}}$ could force moves of wm that were worker and manager moves simultaneously.)

Let WM be an unrestricted IWIM system. Then we define $WM^{\#} = \{wm \in WM \mid wm$ has a nontrivial manager facet$\}$.

A configuration (sts, qs) of an unrestricted IWIM system consists of:

(1) a set $sts = \{(a_{wm}, m_{wm}) \mid wm \in WM\}$ of states (a_{wm}, m_{wm}) one for each automaton in WM;

(2) a set $qs = \{c{:}q_c \mid c \in C_{m_{wm}}, \exists\, a \bullet (a_{wm}, m_{wm}) \in sts\}$ of queues of messages $c{:}[u_0, u_1, \ldots]$ one for each channel $c \in C_{m_{wm}}$ of each management state m_{wm} of each nontrivial manager facet $man(wm)$.

As before, these configuration components are really the ranges of suitable functions.

A configuration (sts, qs) of an unrestricted IWIM system WM is initial iff: all states in sts are initial in both facets, and all channel queues in qs are empty.

Let (sts, qs) be a configuration of an unrestricted IWIM system WM. Then we can define the manager part of (sts, qs) to be $\pi_{\text{man}}(sts) = \{m_{wm} \mid \exists\, a_{wm} \bullet (a_{wm}, m_{wm}) \in sts,\ wm \in WM^{\#}\}$.

A transition of an unrestricted IWIM system WM in configuration (sts, qs) is one of six kinds, patterned after elementary IWIM system transitions:

(ENVI) The environment adds a value to the end of an external input queue.

$$\frac{\begin{array}{l} c \notin \bigcup\{\text{dom}(s_{m'_{wm'}}) \mid m'_{wm'} \in \pi_{\text{man}}(sts)\}\ , \\ c \in \text{dom}(t_{m_{wm}})\ ,\ m_{wm} \in \pi_{\text{man}}(sts)\ , \\ qs_{\text{rest}} = qs - \{c\text{:}[\ \dots\ , u_n]\} \end{array}}{\begin{array}{l} sts \longrightarrow sts\ , \\ qs \longrightarrow qs_{\text{rest}} \cup \{c\text{:}[\ \dots\ , u_n, u]\} \end{array}}$$

(ENVO) The environment removes a value from the end of an external output queue.

$$\frac{\begin{array}{l} c \notin \bigcup\{\text{dom}(t_{m'_{wm'}}) \mid m'_{wm'} \in \pi_{\text{man}}(sts)\}\ , \\ c \in \text{dom}(s_{m_{wm}})\ ,\ m_{wm} \in \pi_{\text{man}}(sts)\ , \\ qs_{\text{rest}} = qs - \{c\text{:}[u, u_1, \dots\ ]\} \end{array}}{\begin{array}{l} sts \longrightarrow sts\ , \\ qs \longrightarrow qs_{\text{rest}} \cup \{c\text{:}[u_1, \dots\ ]\} \end{array}}$$

(IN) A worker facet of an automaton performs an input on one of its input ports, of which there must be at least one.

$$\frac{\begin{array}{l} k\hat{\ }m_{wm}\ ,\ m_{wm} \in \pi_{\text{man}}(sts)\ , \\ (a_k, n_k) \in sts\ ,\ (a_k, n_k) \xrightarrow{i?u} (b_k, n_k)\ , \\ \lambda_{m_{wm}}(p) = i \in I_{wor(k)}\ ,\ t_{m_{wm}}(c) = p\ , \\ sts_{\text{rest}} = sts - \{(a_k, n_k)\}\ , \\ qs_{\text{rest}} = qs - \{c\text{:}[u, u_1, \dots\ ]\} \end{array}}{\begin{array}{l} sts \longrightarrow sts_{\text{rest}} \cup \{(b_k, n_k)\}\ , \\ qs \longrightarrow qs_{\text{rest}} \cup \{c\text{:}[u_1, \dots\ ]\} \end{array}}$$

(OUT) A worker facet of an automaton performs an output on one of its output ports, of which there must be at least one.

$$\frac{\begin{array}{l} (a_k, n_k) \in sts\ ,\ (a_k, n_k) \xrightarrow{o!u} (b_k, n_k)\ , \\ \varnothing \neq Out = \{d \mid \exists\, m_{wm} \in \pi_{\text{man}}(sts), p \bullet k\hat{\ }m_{wm}, \\ \qquad\qquad \lambda_{m_{wm}}(p) = o \in O_{wor(k)}, s_{m_{wm}}(d) = p\}\ , \\ sts_{\text{rest}} = sts - \{(a_k, n_k)\}\ , \\ qs_{\text{rest}} = qs - \{d\text{:}[\ \dots\ , u_{d,n_d}] \mid d \in Out\} \end{array}}{\begin{array}{l} sts \longrightarrow sts_{\text{rest}} \cup \{(b_k, n_k)\}\ , \\ qs \longrightarrow qs_{\text{rest}} \cup \{d\text{:}[\ \dots\ , u_{d,n_d}, u] \mid d \in Out\} \end{array}}$$

(FOR) A port performs a forwarding action.

$$\frac{\begin{array}{l} k\wedge m'_{wm'} \,,\; m'_{wm'} \in \pi_{\mathrm{man}}(sts) \,,\; t_{m'_{wm'}}(c) = p \,, \\ \varnothing \neq Out = \{d \mid \exists\, m_{wm} \in \pi_{\mathrm{man}}(sts),\, p \bullet k\wedge m_{wm}, \\ \qquad\qquad \lambda_{m_{wm}}(p) = o \in O_{wor(k)},\, s_{m_{wm}}(d) = p\} \,, \\ qs_{\mathrm{rest}} = qs - (\{c{:}[u, u_1, \dots]\} \cup \{d{:}[\, \dots , u_{d,n_d}] \mid d \in Out\}) \end{array}}{\begin{array}{l} sts \longrightarrow sts \,, \\ qs \longrightarrow qs_{\mathrm{rest}} \cup \{c{:}[u_1, \dots]\} \cup \{d{:}[\, \dots , u_{d,n_d}, u] \mid d \in Out\} \end{array}}$$

NB. The above notation is intended to include the case that $c \in Out$, whereupon the front message of c's queue is moved to its tail.

(REC) The worker facet of automaton k_r performs a *rec* action a_{k_r} *-rec->* b_{k_r}, moving to state b_{k_r}, and provoking reconfigurations of all the elementary IWIM subsystems managed by manager facets above a current state of which k_r sits. All these manager facets move to their respective new management states. The queues of the channels managed by these manager facets are mapped via the channel reconfiguration data for their particular manager facet.

$$\frac{\begin{array}{l} \varnothing \neq Rm_{\mathrm{man}} = \{m_{wm} \mid m_{wm} \in \pi_{\mathrm{man}}(sts) \bullet k_r\wedge m_{wm}\} \,, \\ (a_{k_r}, m_{k_r}) \in sts \,,\; (a_{k_r}, m_{k_r}) \text{ -}rec\text{-}\!\!> (b_{k_r}, m_{k_r}) \,, \\ Rn_{\mathrm{man}} = \{n_{wm} \mid m_{wm} \in \pi_{\mathrm{man}}(sts) \bullet k_r\wedge m_{wm}, \\ \qquad\qquad r_{k_r\wedge m_{wm}}(rec) = m_{wm} \text{ -}r\text{-}\!\!> n_{wm} = \chi_{m_{wm},n_{wm}} : C_{m_{wm}} \to C_{n_{wm}}\} \,, \\ sts_{\mathrm{rest}} = sts - \;(\{(a_{k_r}, m_{k_r})\} \cup \\ \qquad\qquad \{(a_{wm}, m_{wm}) \mid (a_{wm}, m_{wm}) \in sts,\, m_{wm} \in Rm_{\mathrm{man}}\}) \,, \\ sts_{\mathrm{post}} = \{(b_{k_r}, m_{k_r})\} \cup \{(a_{wm}, n_{wm}) \mid (a_{wm}, m_{wm}) \in sts, \\ \qquad\qquad\qquad\qquad m_{wm} \in Rm_{\mathrm{man}},\, n_{wm} \in Rn_{\mathrm{man}}\} \,, \\ qs_{\mathrm{del}} = \{c{:}q_c \mid c \in C_{m_{wm}},\, c \in \mathrm{dom}(\chi_{m_{wm},n_{wm}}),\, m_{wm} \in Rm_{\mathrm{man}}\} \cup \\ \qquad \{d{:}q_d \mid d \in C_{m_{wm}},\, d \in \mathrm{rng}(\chi_{m_{wm},n_{wm}}),\, m_{wm} \in Rm_{\mathrm{man}},\, n_{wm} \in Rn_{\mathrm{man}}\} \,, \\ qs_{\mathrm{rest}} = qs - qs_{\mathrm{del}} \,, \\ qs_{\mathrm{dom}} = \{c{:}[] \mid c \in C_{m_{wm}},\, c \in \mathrm{dom}(\chi_{m_{wm},n_{wm}}),\, m_{wm} \in Rm_{\mathrm{man}}\} \,, \\ qs_{\mathrm{merge}} = \{d{:}q_{cd} \mid c{:}q_c,\, c \in C_{m_{wm}},\, c \in \mathrm{dom}(\chi_{m_{wm},n_{wm}}), \\ \qquad\qquad d{:}q_d,\, \chi_{m_{wm},n_{wm}}(c) = d \in C_{m_{wm}},\, d \in \mathrm{rng}(\chi_{m_{wm},n_{wm}}), \\ \qquad\qquad m_{wm} \in Rm_{\mathrm{man}},\, n_{wm} \in Rn_{\mathrm{man}}, \\ \qquad\qquad q_{cd} \in \mathrm{merge}(q_c, q_d)\} \end{array}}{\begin{array}{l} sts \longrightarrow sts_{\mathrm{rest}} \cup sts_{\mathrm{post}} \,, \\ qs \longrightarrow qs_{\mathrm{rest}} \cup qs_{\mathrm{dom}} \cup qs_{\mathrm{merge}} \end{array}}$$

The remarks made following the elementary IWIM subsystems transition system description apply with equal or greater force here. Thus all transitions have hypotheses that ensure that any active worker is being actively managed by being above at least one current mangement state. Also there is no murder, only anasthesia and suicide. Moreover, reconfiguration events simultaneously affect all mangers who might be managing a particular worker facet. The structure of the model ensures that they can all do this without adversely interfering with each other.

Let *Confs*(*WM*) be the set of all configurations of *WM*. Equipping it with the transitions just described makes it into a transition system.

A run of *WM* is a sequence of contiguous transitions of *Confs*(*WM*) starting with an initial configuration:

$$(sts, qs) \longrightarrow (sts', qs') \longrightarrow (sts'', qs'') \longrightarrow \ldots$$

Let (sts, qs) be a configuration of *WM*. Let *Mngrs*(*WM*) be the set of manager parts of configurations in *Confs*(*WM*). It can be equipped with transitions derived from those of *Confs*(*WM*). Thus whenever $(sts, qs) \longrightarrow (sts', qs')$ is a (REC) transition of *Confs*(*WM*), there is a *Mngrs*(*WM*) transition $\pi_{\mathrm{man}}(sts) \longrightarrow \pi_{\mathrm{man}}(sts')$. We also add an identity transition $\pi_{\mathrm{man}}(sts) \longrightarrow \pi_{\mathrm{man}}(sts)$ to each manager part in *Mngrs*(*WM*). As previously, all of these transitions are unlabelled.

It will now not be surprising that despite the greater complexity we have here, the projection that we had in [Section 3.1] can be recovered.

Proposition 3.8 Let *WM* be an unrestricted IWIM system. Let *Confs*(*WM*) be the associated transition system, and *Mngrs*(*WM*) be the associated manager parts transition system. Then there is a projection:

$$\Pi : Confs(WM) \to Mngrs(WM)$$

which maps states by:

$$(sts, qs) \longmapsto \pi_{\mathrm{man}}(sts)$$

and which maps (REC) transitions by:

$$\begin{array}{l} (sts, qs) \longrightarrow (sts', qs') \\ \qquad \longmapsto \\ \pi_{\mathrm{man}}(sts) \longrightarrow \pi_{\mathrm{man}}(sts') \end{array}$$

and which maps (ENVI), (ENVO), (IN), (OUT), transitions to identity transitions:

$$\begin{array}{l} (sts, qs) \longrightarrow (sts', qs') \\ \qquad \longmapsto \\ \pi_{\mathrm{man}}(sts) \longrightarrow \pi_{\mathrm{man}}(sts') = \pi_{\mathrm{man}}(sts) \end{array}$$

Proof. Obvious. ☺

Having covered the technical details, it is appropriate to review how the formal constructions relate to the informal account of [Section 2]. As well as the internal details of both manager and worker automata, we have the ^ relation, the λ bijections, and the *r* reconfiguration mappings. Given a worker-manager automaton *wm*, the domains and ranges of ^, λ, *r*, suitably restricted to *wm*, make precise within our model the notion of the environment of *wm* loosely refered to at the beginning of [Section 2]. That these aspects of the model reside outside of the worker and manager facets, reflects the IWIM philosophy that on the one hand workers should be unaware of who they are communicating with or who is in charge of the distributed computation, and that on the other hand managers should have no detailed knowledge of the state of their subordinate workers. For this to work, we need the managers to be ready at all times to react to reconfiguration events from their workers, and if a manager's worker facet is also busy working for his own boss, the asynchronous product between the two facets gives the simplest possible model of the required interruptibility.

In the remainder of the paper we will be concerned only with unrestricted IWIM systems, and will henceforth just refer to them as IWIM systems.

4 IWIM Systems with Delayed Reconfigurations

Now we tackle the problem of the asynchronous nature of true IWIM system event processing. As noted previously, this can be captured within our framework. The basic idea is simple. We introduce fresh pure worker automata, delay automata, whose job is to buffer the reconfiguration events generated by the worker facets of the automata of the original model on their way to the relevant destination manager facet. The way this is done is to change the *rec* events of the original model into *rec* messages to the delay automata, who then subsequently raise the required event. Since buffering is already implicit in the message queues used by worker facets, and further buffering can be achieved by retaining information in automaton states, there are a number of ways one can imagine of implementing such an idea. In the one we will follow, the workers each acquire an extra output port through which to send *rec* messages instead of raising *rec* events. Connected to these extra output ports, are channels leading to delay automata, one per manager facet in charge of the worker. This ensures that the *rec* messages are broadcast asynchronously towards each relevant manager. (Because event processing takes place simultaneously by all managers below a worker, we need to ensure that each delay automaton is above only one manager. To ensure the correct separation of concerns between automata it is easiest to introduce delay automata on a per per $wm' \wedge m_{wm}$ tuple basis.) Upon receipt of the *rec* message, the delay automaton raises the corresponding event with the manager.

Assuming that some particular worker facet is above k manager facets, the behaviour of the original system can be recovered as long as there is always the possibility of performing the following $2k+1$ step sequence of the new system instead of a *rec* transition of the original system, in a manner uninterrupted by other system transitions:

(1) the worker facet transmits the relevant *rec* value through its extra output port onto the n delay channels leading to the n delay automata corresponding to the n manager facets above which it sits,

(2_i) delay automaton i receives the *rec* value from delay channel i, recording it in its state,

(3_i) delay automaton i performs a *rec* transition causing manager facet i to perform the required reconfiguration.

This sequence of steps preserves the property that all delay channels remain empty except between steps (1) and (2_i), which is correspondingly consistent with enabling them to be executed without interruptions.

On the other hand, if we consider that the execution of these steps can indeed be interrupted, as allowed by the asynchrony inherent in the fragmenting of a single transition into several, other outcomes become possible. Since the original system had only synchronous reconfigurations, it provides no definition of what might happen should a reconfiguration be attempted nonatomically, and any evolution consistent with the semantics is permissible. For example, a context dependent notion of reconfiguration can be created by having delay automata raise different reconfiguration actions in manager facets, depending on what reconfigurations intervened between the receiving of some particular *rec* value from a worker, and the raising of the corresponding reconfiguration event in the manager; the information to manage this being kept in a delay automaton's state, suitably managed through intervening reconfigurations. And depending on what policy is adopted for the introduction and behaviour of the delay automata, different policies for the handling of pending events become possible. Moreover being themselves workers, delay automata can be woken and suspended during reconfiguration transitions, further tuning this aspect.

One canonical possibility for dealing with reconfigurations that attempt to interleave other reconfiguration actions, is to enforce a strict sequentialisation policy. This can be done by ensuring that once a *rec* message arrives at a delay automaton, the only thing the delay automaton can then do is to raise the corresponding event, ignoring further inputs till it has done so. We call this arrangement the standard asynchronisation of an IWIM system, and we now present the technical details.

Suppose *WM* is an IWIM system with the usual notations, i.e. the typical automaton name is *wm* mapping to $(I, O, A = (St, Init, Tr))\otimes(M, m_{\mathrm{I}}, R)$, with manager states m mapping to networks $(P_{m_{wm}}, C_{m_{wm}})$, and reconfigurations m_{wm} -*r*-› $n_{wm} = \chi_{m_{wm},n_{wm}} : C_{m_{wm}} \rightarrow C_{n_{wm}}$; and with ancillary data given by $wm'{}^\wedge m_{wm}$, $\lambda_{m_{wm}}$, $r_{wm'{}^\wedge m_{wm}}$.

The standard asynchronisation of *WM*, which we call here *WM**, has the set of automaton names $WM^* = WM \cup \{\Delta.wm'.m.wm \mid wm'{}^\wedge m_{wm}\}$. We assume all of these $\Delta.wm'.m.wm$ names are fresh, and introduce for each $\Delta.wm'.m.wm$ name, for future convenience, fresh port, channel, and input and output port names[2]:

$$\Delta.wm'.m.wm_{\mathsf{s}}\ ,\ \Delta.wm'.m.wm_{\mathsf{t}}\ ,\ \Delta.wm'.m.wm_{\mathsf{ch}}\ ,\ \Delta.wm'.m.wm_{\mathsf{i}}\ ,\ \Delta.wm'_{\mathsf{o}}$$

If *wm* maps to $(I, O, A = (St, Init, Tr))\otimes(M, m_{\mathrm{I}}, R)$ in *WM*, in *WM**, *wm* maps to $(I, O^*, A^* = (St, Init, Tr^*))\otimes(M, m_{\mathrm{I}}, R^*)$.

The input ports I of the worker facet of *wm* remain unchanged. However for the output ports we have $O^* = O \cup \{\Delta.wm_{\mathsf{o}}\}$. The worker facet automaton *wor*(*wm*) itself is given by the same state space *St*, initial state *Init*, and:

$$Tr^* = Tr_{\mathrm{I}} \cup Tr_{\mathrm{O}} \cup \{a \text{ -}\Delta.wm_{\mathsf{o}}!rec\text{-› } b \mid a \text{ -}rec\text{-› } b \in Tr_{\mathrm{R}}\}$$

This ensures that *rec* messages can be sent over $\Delta.wm_{\mathsf{o}}$ to all delay automata $\Delta.wm.m'.wm'$. To ensure that these are handled properly, we examine the manager facet of *wm*.

In the manager facet *man*(*wm*), the state space M and initial state m_{I} remain unchanged. State m however maps to the communication network $(P^*_{m_{wm}}, C^*_{m_{wm}})$ where:

$$P^*_{m_{wm}} = P_{m_{wm}} \cup \{\Delta.wm'.m.wm_{\mathsf{s}}, \Delta.wm'.m.wm_{\mathsf{t}} \mid wm'{}^\wedge m_{wm}\}$$
$$C^*_{m_{wm}} = C_{m_{wm}} \cup \{\Delta.wm'.m.wm_{\mathsf{ch}} \mid wm'{}^\wedge m_{wm}\}$$

$$s^*_{m_{wm}} = s_{m_{wm}} \cup \{\Delta.wm'.m.wm_{\mathsf{ch}} \mapsto \Delta.wm'.m.wm_{\mathsf{s}} \mid wm'{}^\wedge m_{wm}\}$$
$$t^*_{m_{wm}} = t_{m_{wm}} \cup \{\Delta.wm'.m.wm_{\mathsf{ch}} \mapsto \Delta.wm'.m.wm_{\mathsf{t}} \mid wm'{}^\wedge m_{wm}\}$$

2. The last of these is not an error.

Finally, if m_{wm} -r-› n_{wm} = $\chi_{m_{wm},n_{wm}} : C_{m_{wm}} \to C_{n_{wm}}$ is a reconfiguration transition of R, there is a corresponding transition of R^* given by $\chi^*_{m_{wm},n_{wm}} : C^*_{m_{wm}} \to C^*_{n_{wm}}$ where $\chi^*_{m_{wm},n_{wm}} = \chi_{m_{wm},n_{wm}}$ interpreted as a partial injection on $C^*_{m_{wm}}$.

Standing between the worker and manager facets of the preceding automata, are the delay automata themselves. A delay automaton name $\Delta.wm'.m.wm$ maps to a pure worker given by:

$$(I_{\Delta.wm'.m.wm}, O_{\Delta.wm'.m.wm}, A_{\Delta.wm'.m.wm} = \\ (St_{\Delta.wm'.m.wm}, Init_{\Delta.wm'.m.wm}, Tr_{\Delta.wm'.m.wm}))\otimes(\{\bullet\}, \bullet, \varnothing)$$

Here:

$$I_{\Delta.wm'.m.wm} = \{\Delta.wm'.m.wm_{\mathrm{i}} \mid wm' \wedge m_{wm}\}$$

while $O_{\Delta.wm'.m.wm} = \varnothing$. The worker automaton $A_{\Delta.wm'.m.wm}$ is given by the state space:

$$St_{\Delta.wm'.m.wm} = Rec_{wm'} \uplus \{Init_{\Delta.wm'.m.wm}\}$$

and the initial state $Init_{\Delta.wm'.m.wm}$ is the one named as such. The transitions of $A_{\Delta.wm'.m.wm}$ are given by:

$$Tr_{\Delta.wm'.m.wm} = \{Init_{\Delta.wm'.m.wm} \text{ -}\Delta.wm'.m.wm_{\mathrm{i}}?rec\text{-› } rec \mid rec \in Rec_{wm'}\} \cup \\ \{rec \text{ -}rec\text{-› } Init_{\Delta.wm'.m.wm} \mid rec \in Rec_{wm'}\}$$

where we have abused notation a little by allowing *rec* to name the state reached by inputting a *rec* message (not to mention its original use as event name), hopefully without causing confusion. It is now clear that the delay automaton inputs a *rec* message coming from the original worker, and then provokes a *rec* reconfiguration event in the manager at a later point.

To connect all this together, we give the above relation, which is:

$$\wedge^* = \wedge \cup \{\Delta.wm'.m.wm \wedge^* m_{wm} \mid wm' \wedge m_{wm}\}$$

and the $\lambda^*_{m_{wm}}$ bijections which are:

$$\lambda^*_{m_{wm}} = \lambda_{m_{wm}} \cup \{\Delta.wm'.m.wm_{\mathrm{s}} \mapsto \Delta.wm'_{0} \mid wm' \wedge m_{wm}\} \cup \\ \{\Delta.wm'.m.wm_{\mathrm{t}} \mapsto \Delta.wm'.m.wm_{\mathrm{i}} \mid wm' \wedge m_{wm}\}$$

Note how in the first line of the above the original worker's output port $\Delta.wm'_0$ is shared by as many managers as it has, each controlling an individual queue to a separate $\Delta.wm'.m.wm$ delay automaton.

Finally the $r^*_{\Delta.wm'.m.wm \wedge^* m_{wm}}$ functions are given by:

$$r^*_{\Delta.wm'.m.wm \wedge^* m_{wm}}(rec) = m_{wm} \text{ -}r\text{-› } n_{wm} \;\text{ iff }\; r_{wm' \wedge m_{wm}}(rec) = m_{wm} \text{ -}r\text{-› } n_{wm}.$$

It is now clear that this construction has the properties indicated informally above. Thus whereas in *WM*, a worker *wm′* above a manger state m_{wm} can perform the step *a* -*rec*-› *b* simultaneously with each implicated manager's performing the appropriate m_{wm} -r-› n_{wm} (because $r_{wm' \wedge m_{wm}}$ maps *rec* to m_{wm} -r-› n_{wm}), in *WM**, *wm′* can no longer do this directly. Instead it passes a *rec* message to $\Delta.wm'.m.wm$ via a single *a* -$\Delta.wm'_0!rec$-› *b* action which causes *rec* messages to be broadcast onto all relevant channels $\Delta.wm'.m.wm_{\mathrm{ch}}$. If such a channel was previously empty, then $\Delta.wm'.m.wm$ can swallow the *rec* message by performing an $Init_{\Delta.wm'.m.wm}$ -$\Delta.wm'.m.wm_{\mathsf{i}}?rec$-› *rec* input from the same channel. This obtains by the fact that ports $\Delta.wm'_{\mathsf{0}}$ and $\Delta.wm'.m.wm_{\mathsf{i}}$ are connected via $\Delta.wm'.m.wm_{\mathsf{ch}}$, since $\lambda*_{m_{wm}}$ connects $\Delta.wm'_{\mathsf{0}}$ to $\Delta.wm'.m.wm_{\mathsf{s}} = s*_{m_{wm}}(\Delta.wm'.m.wm_{\mathsf{ch}})$, and also connects $t*_{m_{wm}}(\Delta.wm'.m.wm_{\mathsf{ch}}) = \Delta.wm'.m.wm_{\mathsf{t}}$ to $\Delta.wm'.m.wm_{\mathsf{i}}$. Since $r*_{\Delta.wm'.m.wm\wedge*m_{wm}}$ maps the only available $\Delta.wm'.m.wm$ transition *rec* -*rec*-› $Init_{\Delta.wm'.m.wm}$ to the reconfiguration m_{wm} -*r*-› n_{wm}, it follows that when $\Delta.wm'.m.wm$ performs *rec* -*rec*-› $Init_{\Delta.wm'.m.wm}$, it provokes the desired reconfiguration m_{wm} -*r*-› n_{wm}. Thus if $\Delta.wm'.m.wm_{\mathsf{ch}}$ was empty at the outset, the simulation of one manager's reconfiguration by a delayed but uninterrupted sequence of steps is available. Evidently when several managers need to react, consequent on the same original atomic reconfiguration, similar simulations can also be constructed. These simulations may also be interleaved with other actions, provided none of the other actions 'beat the sequence to the tape', where the 'tape' is the invocation of a *rec* step mapped by a $r*_{\Delta.wm'.m.wm\wedge*m_{wm}}$ to a change of configuration of the manager *wm*, while the manager remains in the original state *m*. Examples of other actions that can safely be interleaved in this manner are ordinary I/O actions, and reconfigurations not involving any of the automata involved.

Proposition 4.1 The construction just given is idempotent, in the sense that applying it *n* more times to *WM** results in a system $WM\underbrace{*\cdots*}_{n+1}$ which can simulate an atomic reconfiguration of *WM* that involves *k* managers in $2k(n+1)+1$ uninterrupted steps.

The straightforward if tedious proof rests on the observation that in *WM**, the only worker above m_{wm} capable of provoking a reconfiguration is a $\Delta.wm'.m.wm$, so that the next application of the construction replaces each $\Delta.wm'.m.wm$'s *rec* steps by a three step sequence etc. Thus iterated application of the construction exemplifies the fact that a chain of buffers is behaviourally equivalent to a single buffer.

5 The Arbab, de Boer, Bonsangue Model

In this section we show how the model proposed by Arbab, de Boer and Bonsangue in [Arbab et al. (2000a)] (see also [Arbab et al. (2000b)]), henceforth the ABB model, can be subsumed within our framework. In the ABB model, there is a family of *components*. Each component is a transition system similar to one of our worker automata, and it has access to a set of channel ends to which it is connected. A component may output values along channel source ends (eg. $\overline{c}$) to which it is connected, and may input values from channel sink ends (eg. *c*) to which it is connected. The state transitions for these actions are of the form a -$\overline{c}!v$-› b and a -$c?v$-› b respectively, and these are the only kinds of action that components may perform. The dynamic reconfigurability of ABB systems comes from the fact that they can alter their set of connected channel ends by sending and receiving channel end identities along the channels themselves. Thus if a component possesses channel ends $\overline{c}$, *d*, it may relinquish possession of *d* by a transition like a -$\overline{c}!d$-› b; likewise a -$\overline{c}!\overline{d}$-› b relinquishes possession of $\overline{d}$. Likewise possession of *d* or $\overline{d}$ can be gained by a -$c?d$-› b or a -$c?\overline{d}$-› b. It is tacitly assumed that since channels are point to point connections, once a component has relinquished possession of a channel end, it will no longer attempt to use it until it has received it once again from some other component. Channels themselves are queues in the ABB model, just as they are in ours, and when a channel end, d (resp. $\overline{d}$) say, becomes detached from the component to which it was previously connected by being output along channel c say, no inputs over d (resp. outputs over $\overline{d}$) can take place until the relevant message has been consumed by the component connected to the sink end of c, whereupon d (resp. $\overline{d}$) becomes available to that component for communication purposes. Output and input transitions in which a channel end is respectively transmitted or received are called reconfiguring output and input transitions.

We will now describe the mapping of a family of ABB components to a corresponding IWIM system. Note that since channels are not created dynamically in the ABB model, the complete set of channels that figure in an execution of an ABB system is known at initialisation time, and given an ABB system, we call this complete set of channels CH. From this we create the five disjoint alphabets:

$CH_{\mathrm{i}} = \{ch_{\mathrm{i}} \mid ch \in CH\}$
$CH_{\mathrm{o}} = \{ch_{\mathrm{o}} \mid ch \in CH\}$
$CH_{\mathrm{s}} = \{ch_{\mathrm{s}} \mid ch \in CH\}$
$CH_{\mathrm{t}} = \{ch_{\mathrm{t}} \mid ch \in CH\}$
$CH_{\mathrm{ch}} = \{ch_{\mathrm{ch}} \mid ch \in CH\}$

Let $C_1 \dots C_n$ be a family of ABB components. For each C_i we construct a transition system K_i as follows. Let C_i be $(St_i, Init_i, Tr_i, r_i)$ where St_i is a set of states of which $Init_i$ is an initial state, Tr_i is a transition relation containing transitions of type a -$\overline{out}!v$-› b or a -$in?v$-› b (with $in, out \in CH$), and r_i is the initial value of the dynamically changing set of channel ends possessed by C_i. By the remarks above we can assume that $CH = \{ch \mid$ for some i, $ch \in r_i$ or $\overline{ch} \in r_i\}$. For simplicity we will assume that each end of each channel in CH is in some r_i.

Now we set K_i to be the transition system given by $(St_i*, Init_i*, Tr_i*)$, where the set of states is $St_i* = St_i \cup newSt_i$, with $Init_i* = Init_i$, and Tr_i* is given as follows (also implicitly defining the fresh states $newSt_i$). Each transition a -$\overline{out}!v$-› b or a -$in?v$-› b of C_i where v is not a channel end yields a transition a -$out_{\mathrm{o}}!v$-› b or a -$in_{\mathrm{i}}?v$-› b of K_i. Moreover each reconfiguring output a -$\overline{out}!\overline{ch}$-› b of C_i is replaced by a pair of transitions a -$out_{\mathrm{o}}!ch_{\mathrm{o}}$-› ab -$rec(out_{\mathrm{o}}!ch_{\mathrm{o}})$-› b, where ab is a fresh state in $newSt_i$ and $rec(out_{\mathrm{o}}!ch_{\mathrm{o}})$ is a reconfiguration action where the intention is to simulate the detaching of the channel end ch_{o} from the component in a manner that will be made clear below. Likewise if the channel end being detached is ch rather than $\overline{ch}$, K_i will contain the pair of transitions a -$out_{\mathrm{o}}!ch_{\mathrm{i}}$-› ab -$rec(out_{\mathrm{o}}!ch_{\mathrm{i}})$-› b. A similar arrangement holds for reconfiguring input transitions a -$in?\overline{ch}$-› b and a -$in?ch$-› b. For these we have respectively a -$in_{\mathrm{i}}?ch_{\mathrm{o}}$-› ab -$rec(in_{\mathrm{i}}?ch_{\mathrm{o}})$-› b and a -$in_{\mathrm{i}}?ch_{\mathrm{i}}$-› ab -$rec(in_{\mathrm{i}}?ch_{\mathrm{i}})$-› b.

For technical reasons, it is not sufficient to work with just the K_i. Given K_i, let θ_i^{+a} be a finite directed path through the transition system of K_i (i.e. a finite sequence of contiguous transitions of K_i), starting at state a. Let K_i^a be the transition system determined by the set of paths: $\{\theta_i^{+a} \mid \theta_i^{+a}$ is a path through the transition system of K_i starting at a, and if θ_i^{+a} contains a rec transition, there is only one and it is the last transition of $\theta_i^{+a}\}$.

Given a θ_i^{+a}, let θ_i^a be the result of erasing from θ_i^{+a} all non-rec transitions (so the transitions listed in θ_i^a will not be contiguous, neither will they necessarily mention a).

Let $\phi(\theta_i^{+a})$, $\phi(\theta_i^{a})$ denote the final state reached by such a θ_i^{+a} or θ_i^{a}. Define $\Theta_i^{a} = \{\theta_i^{a} \mid \theta_i^{+a}$ is a path through the transition system of K_i starting at $a\}$; consequently Θ_i^{a} is partially ordered by the prefix relation. We write θ_i^{+}, θ_i, Θ_i to denote $\theta_i^{+Init_i}$, $\theta_i^{Init_i}$, $\Theta_i^{Init_i}$. Let:

$$M = \prod\{\Theta_i \mid i \in \{1 \dots n\}\}$$

The rest of the construction will proceed by recursion on the structure of M, which is again partially ordered by the prefix relation. We construct a pure manger automaton *pm*, whose space of states is M, and above each $m \in M$, there will be a collection of pure worker automata crafted from the K_i^{a} transition systems[3].

The base case is $m = [\,]\times[\,]\times\dots\times[\,]$. Above this m we have the collection of pure workers $pw_i^{[\,]}$ for $i \in \{1 \dots n\}$, where $pw_i^{[\,]}$ is given by $(CH_{\mathrm{i}i}^{[\,]}, CH_{0i}^{[\,]}, K_i^{Init_i})$, with $CH_{\mathrm{i}i}^{[\,]} = \{ch_{\mathrm{i}} \mid ch_{\mathrm{i}} \in CH_{\mathrm{i}}, ch \in r_i\}$ and $CH_{0i}^{[\,]} = \{ch_0 \mid ch_0 \in CH_0, \overline{ch} \in r_i\}$. Note that $Init_i = \phi([\,])$ (with the understanding that $[\,]$ is the empty path through K_i).

The manager state m maps to (P_m, C_m) where:

$$P_m = \{ch_{\mathrm{s}} \mid ch_{\mathrm{s}} \in CH_{\mathrm{s}}, \overline{ch} \in r_i\} \cup \{ch_{\mathrm{t}} \mid ch_{\mathrm{t}} \in CH_{\mathrm{t}}, ch \in r_i\}$$
$$C_m = \{ch_{\mathrm{ch}} \mid \{ch_{\mathrm{s}}, ch_{\mathrm{t}}\} \cap P_m \neq \varnothing\}$$

and the s_m, t_m maps function in the way we would expect, i.e. $s_m(ch_{\mathrm{ch}}) = ch_{\mathrm{s}}$ and $t_m(ch_{\mathrm{ch}}) = ch_{\mathrm{t}}$. The link between the manager and the workers is also unsurprising:

$$\lambda_m = \{ch_{\mathrm{t}} \mapsto ch_{\mathrm{i}} \mid ch_{\mathrm{i}} \in CH_{\mathrm{i}i}^{[\,]}\} \cup \{ch_{\mathrm{s}} \mapsto ch_0 \mid ch_0 \in CH_{0i}^{[\,]}\}$$

$$pw_i^{[\,]}\wedge m$$

completing the base case.

Now suppose that $m = (\theta_1 \dots \theta_n)$ and suppose $m' = (\theta_1 \dots \theta_i' \dots \theta_n)$ where $\theta_i' = \theta_i @[a_i \text{-}rec(out_0!ch_0)\text{-›}\, b_i]$, and where the transition $a_i \text{-}rec(out_0!ch_0)\text{-›}\, b_i$ is a K_i- immediate successor reconfiguring transition to the last one in θ_i. The manager state m which maps to (P_m, C_m) is transformed to m' which maps to $(P_{m'}, C_{m'})$ where:

$$P_{m'} = P_m - \{ch_{\mathrm{s}}\}$$
$$C_{m'} = \{ch_{\mathrm{ch}} \mid \{ch_{\mathrm{s}}, ch_{\mathrm{t}}\} \cap P_{m'} \neq \varnothing\}$$

and the $s_{m'}$, $t_{m'}$ maps work as expected, i.e. $s_{m'}(ch_{\mathrm{ch}}) = ch_{\mathrm{s}}$ and $t_{m'}(ch_{\mathrm{ch}}) = ch_{\mathrm{t}}$. It now makes sense to define the manager reconfiguration transition $m \text{-}r\text{-›}\, m'$ as the partial injection

$$\chi_{m,m'} : C_m \to C_{m'}$$

which is the maximal identity function on $C_m \cap C_{m'}$.

Suppose that above m we had the n pure workers $\{pw_j^{\theta_j} \mid j \in \{1 \dots n\}\}$. Then above m' we will also have n pure workers. For $j \neq i$, $pw_j^{\theta_j}$ will continue to be above m' and the reconfiguration transition $m \text{-}r\text{-›}\, m'$ will leave it in the same state as it was. For the case $j = i$ we have instead the pure worker $pw_i^{\theta_i'} = (CH_{\mathrm{i}i}^{\theta_i'}, CH_{0i}^{\theta_i'}, K_i^{\phi(\theta_i')})$ where:

$$CH_{\mathrm{i}i}^{\theta_i'} = CH_{\mathrm{i}i}^{\theta_i}$$
$$CH_{0i}^{\theta_i'} = CH_{0i}^{\theta_i} - \{ch_0\}$$

3. Since there is only one nontrivial manager, we suppress the '*pm*' tags for convenience.

and so we can summarise the above map for m' as:

$$\{pw_j^{\theta_j} \wedge m' \mid pw_j^{\theta_j} \wedge m,\ j \in \{1 \ldots n\} - \{i\}\} \cup \{pw_i^{\theta_i'} \wedge m'\}$$

The $\lambda_{m'}$ map is:

$$\lambda_{m'} = \lambda_m - \{ch_\mathrm{s} \mapsto ch_0\}$$

and we have that:

$$r_{pw_i^{\theta_i} \wedge m}(rec(out_0!ch_0)) = m \text{ -}r\text{-›}\ m'$$

which completes the piece of the recursion for the case of a $rec(out_0!ch_0)$ reconfiguration. If we consider instead $rec(out_0!ch_\mathrm{i})$, $rec(in_\mathrm{i}?ch_0)$, $rec(in_\mathrm{i}?ch_\mathrm{i})$ reconfigurations, the above is modified respectively by:

$$CH_{\mathrm{i}i}^{\theta_i'} = CH_{\mathrm{i}i}^{\theta_i} - \{ch_\mathrm{i}\} \ ;\ CH_{0i}^{\theta_i'} = CH_{0i}^{\theta_i} \ ;$$
$$P_{m'} = P_m - \{ch_\mathrm{t}\} \ ;\ C_{m'} = \{ch_\mathrm{ch} \mid \{ch_\mathrm{s}, ch_\mathrm{t}\} \cap P_{m'} \neq \varnothing\} \ ;$$
$$\lambda_{m'} = \lambda_m - \{ch_\mathrm{t} \mapsto ch_\mathrm{i}\}$$

$$CH_{\mathrm{i}i}^{\theta_i'} = CH_{\mathrm{i}i}^{\theta_i} \ ;\ CH_{0i}^{\theta_i'} = CH_{0i}^{\theta_i} \cup \{ch_0\} \ ;$$
$$P_{m'} = P_m \cup \{ch_\mathrm{s}\} \ ;\ C_{m'} = \{ch_\mathrm{ch} \mid \{ch_\mathrm{s}, ch_\mathrm{t}\} \cap P_{m'} \neq \varnothing\} \ ;$$
$$\lambda_{m'} = \lambda_m \cup \{ch_\mathrm{s} \mapsto ch_0\}$$

$$CH_{\mathrm{i}i}^{\theta_i'} = CH_{\mathrm{i}i}^{\theta_i} \cup \{ch_\mathrm{i}\} \ ;\ CH_{0i}^{\theta_i'} = CH_{0i}^{\theta_i} \ ;$$
$$P_{m'} = P_m \cup \{ch_\mathrm{t}\} \ ;\ C_{m'} = \{ch_\mathrm{ch} \mid \{ch_\mathrm{s}, ch_\mathrm{t}\} \cap P_{m'} \neq \varnothing\} \ ;$$
$$\lambda_{m'} = \lambda_m \cup \{ch_\mathrm{t} \mapsto ch_\mathrm{i}\}$$

together with the obvious consequences. Since the ABB system enjoys the property that a component cannot give away a channel end that it is not connected to and neither does it ever receive a channel end that it already possesses, it readily follows that the set operations above are nonnull.

Beyond these there are the expected identity transitions on states of M of course, which completes the construction. Thus we have cut up the original ABB system into a collection of pieces that can be reassembled as an IWIM system, in order that the latter is able to achieve the same effect as the original system. In fact it is easy to convince onself that the IWIM system constructed from a given ABB system by the above technique is able to simulate it in the sense that non-reconfiguring inputs and outputs correspond bijectively, while reconfiguring inputs and outputs correspond to sequences of two steps in the IWIM system, the first to receive or transmit the channel end identifier, the second to provoke the desired reconfiguration via the manager.

6 The Katis, Sabadini, Walters Model

In this section we consider a model proposed by Katis, Sabadini and Walters in [Katis et al. (2000)], henceforth the KSW model, and show how it too can be subsumed within our framework. In the KSW model, the main entity of interest is the CP automaton. A CP automaton $\mathcal{G} = (\mathbf{G}, X, Y, A, B, \partial_0, \partial_1, \gamma_0, \gamma_1)$, consists of a directed graph $\mathbf{G} = (G_0, G_1)$ where G_0 is the set of nodes and G_1 is the set of arcs, together with four maps:

$$\partial_0 : G_1 \to X \ ;\ \partial_1 : G_1 \to Y \ ;\ \gamma_0 : A \to G_0 \ ;\ \gamma_1 : B \to G_0$$

These work as follows. The arcs of the graph represent transitions of the automaton, whose states are the nodes. The sets X and Y are input and output alphabets respectively. Thus the maps $\partial_0 : G_1 \to X$ and $\partial_1 : G_1 \to Y$ describe which input letter a transition of the graph consumes, and which output letter it produces. Since both maps are total, each transition involves both input and output. We will write a CP automaton transition as:

s -(*ind*, *arc*, *outd*)-› *t*

where *s* and *t* are states, *arc* is the arc carrying the transition, and *ind*, *outd* are the input and output data. (In [Katis et al. (2000)], the authors also admit null elements in both *X* and *Y* alphabets, to aid abstraction and to represent the absence of genuine communication during a step.) Communication is synchronous, thus when two CP automata communicate, the symbol output by the producer of the communication, is simultaneously input by the consumer of the communication. Most emphatically, there are no queues in the model: communication in this model is above all a synchronisation mechanism.

The sets *A* and *B* (called the in-condition and out-condition respectively in [Katis et al. (2000)]), are to do with initialisation and finalisation, though in a slightly non-standard manner. Specifically, the γ_0-image of *A* is the set of entry points into the CP automaton, i.e. initial states, and the γ_1-image of *B* is the set of exit points, i.e. final states, of the automaton — except that when CP automata are combined in the appropriate way, then subsets of entry or exit points may be identified, leading to a richer gamut of possibilities parameterised by partitions of $\gamma_0(A)$ and $\gamma_1(B)$.

CP automata are endowed with a number of algebraic operations, which construct more complex CP automata out of simpler ones. We will model the KSW formalism by mapping CP automata to IWIM systems, and then showing how the CP automaton algebraic operations can be reflected in constructions on the corresponding IWIM systems.

Let $\mathcal{G} = (\mathbf{G} = (G_0, G_1), X, Y, A, B, \partial_0, \partial_1, \gamma_0, \gamma_1)$ be a CP automaton. We build an IWIM system corresponding to $\mathcal{G}$, and consisting of a pure manager and a pure worker. The pure manager *pm* has one-state $\blacklozenge$ which maps to $(\{p_s, p_t\}, \{ch_s, ch_t\})$ with $s_\blacklozenge(ch_s) = p_s$ and $t_\blacklozenge(ch_t) = p_t$ (and with $s_\blacklozenge(ch_t)$ and $t_\blacklozenge(ch_s)$ undefined). The state $\blacklozenge$ is initial and the only transition of the manager is the identity. Clearly the manager's structure is independent of $\mathcal{G}$.

The pure worker *pw* is $(\{p_i\}, \{p_0\}, (St, Init, Tr))$ where the transition system *Tr* is constructed thus. For each $\mathcal{G}$ transition *s* -(*ind*, *arc*, *outd*)-› *t*, *Tr* contains the two step sequence *s* -p_i?*ind*-› *arc* -p_0!*outd*-› *t* ; this makes it clear that $St = G_0 \cup G_1$ (we will tacitly assume that this union is disjoint). Regarding *Init*, we can choose *any* state s_0 in $\gamma_0(A)$ to be *Init*. Thus the mapping from CP automata to IWIM systems is in general one to many. In reality of course, examples of CP automata that represent complete systems typically have unique initial states, reflecting the often observed fact that most real systems start in a well defined condition. The plurality comes in useful when component CP automata are combined to form a larger system. We will comment on this further below. More generally, $\gamma_0(A)$ and $\gamma_1(B)$ are sets of states of the pure worker *pw*.

Our basic construction is nearly complete. All that remains is to note that the λ mapping is given by:

$$\lambda_\blacklozenge(p_{\mathrm{s}}) = p_{\mathrm{o}} \;\; ; \;\; \lambda_\blacklozenge(p_{\mathrm{t}}) = p_{\mathrm{i}}$$

that the above mapping is given by:

$$pw{}^{\wedge}{}^{\blacklozenge}{}_{pm}$$

and that since there are no *rec* actions in the worker, the *r* map is empty.

Note the following invariant of the generated IWIM system: regardless of G, there is exactly one pure worker, one one-state pure manager, one external input channel, one external output channel, and $\gamma_0(A)$ and $\gamma_1(B)$ can be identified with sets of configurations of the pure worker.

We can easily see that whatever the initial state of the given CP automaton, we can find an IWIM system from among the possibilities constructed, with the same initial state; and which furthermore simulates it in the sense that the execution of a CP automaton transition inputting x and outputting y, corresponds in the IWIM system to the input from the input queue of x and the output onto the output queue of y, in that order. (The alternative order leads to an equally acceptable construction.) Note that in the IWIM system these are comunications with the environment.

We now move on to constructions on CP automata and how these are reflected in the corresponding IWIM systems; the principal ones that we must consider are binary combinators. We will subscript with the name of the relevant automaton to disambiguate when notations would otherwise clash.

Communicating Parallel Composition. Let $G = (\mathbf{G} = (G_0, G_1), X, Y, A, B, \partial_{0,G}, \partial_{1,G}, \gamma_{0,G}, \gamma_{1,G})$ and $H = (\mathbf{H} = (H_0, H_1), Y, Z, C, D, \partial_{0,H}, \partial_{1,H}, \gamma_{0,H}, \gamma_{1,H})$ be CP automata. Then the communicating parallel composition of G and H, written $G \cdot H$, is the CP automaton:

$$\begin{aligned} G \cdot H = (&\mathbf{G}\cdot\mathbf{H} = (G_0 \times H_0, G_1{\cdot}H_1 = \{(g, h) \mid g \in G_1, h \in H_1, \partial_{1,G}(g) = \partial_{0,H}(h)\}),\\ &X, Z,\\ &A \times C, B \times D,\\ &\partial_{0,G \cdot H}(g, h) = \partial_{0,G}(g),\ \ \partial_{1,G \cdot H}(g, h) = \partial_{1,H}(h),\\ &\gamma_{0,G \cdot H} = \gamma_{0,G} \times \gamma_{0,H}\,,\ \ \gamma_{1,G \cdot H} = \gamma_{1,G} \times \gamma_{1,H}\,) \end{aligned}$$

This definition makes clear the statement above that communication is synchronous in the KSW model. The input and output labels on an arc (g, h) of the combined system are $\partial_{0,G}(g)$ and $\partial_{1,H}(h)$ respectively, while the very existence of the arc is predicated on the condition $\partial_{1,G}(g) = \partial_{0,H}(h)$, which supports the interpretation that arc g output and arc h input the same symbol. This is the only notion of communication in the KSW model.

We model the communicating parallel composition of G and H at the IWIM system level as follows. Suppose WM_G is an IWIM system representing G, and WM_H is an IWIM system representing H. We assume that both WM_G and WM_H each have a pure worker, pw_G and pw_H respectively, a one-state pure manager, pm_G and pm_H respectively, an external input channel $ch_{\mathrm{t},G}$ and $ch_{\mathrm{t},H}$ respectively, an external output channel $ch_{\mathrm{s},G}$ and $ch_{\mathrm{s},H}$ respectively, that $\gamma_{0,G}(A)$ and $\gamma_{1,G}(B)$ can be identified with a set of states of

pw_G, and that $\gamma_{0,H}(C)$ and $\gamma_{1,H}(D)$ can be identified with a set of states of pw_H. The IWIM system $WM_{G \cdot H}$ we seek can be generated from WM_G and WM_H as follows.

There is the usual one-state pure manager $pm_{G \cdot H}$ as above. The corresponding pure worker $pw_{G \cdot H} = (\{p_i\}, \{p_0\}, (St_{G \cdot H}, Init_{G \cdot H}, Tr_{G \cdot H}))$ is built from pw_G and pw_H by defining $St_{G \cdot H} = St_G \times St_H$, $Init_{G \cdot H} = (Init_G, Init_H)$, and for $Tr_{G \cdot H}$, whenever we have a pair of transitions in Tr_G of the form s_G -p_i?*ind*-› $arc_{st,G}$ -p_0!*val*-› t_G, and a pair of transitions in Tr_H of the form s_H -p_i?*val*-› $arc_{st,H}$ -p_0!*outd*-› t_H, we form the $Tr_{G \cdot H}$ transitions (s_G, s_H) -p_i?*ind*-› $(arc_{st,G}, arc_{st,H})$ -p_0!*outd*-› (t_G, t_H). It is clear that this procedure only succeeds because of the special structure of the transition systems Tr_G and Tr_H. We can now identify $\gamma_{0,G \cdot H}(A \times C)$ with states corresponding to $\gamma_{0,G}(A) \times \gamma_{0,H}(C)$, and $\gamma_{1,G \cdot H}(B \times D)$ with states corresponding to $\gamma_{1,G}(B) \times \gamma_{1,H}(D)$; and the rest of the data for the IWIM system $WM_{G \cdot H}$ is routine.

It is obvious that $WM_{G \cdot H}$ is able to simulate $G \cdot H$ in a straightforward manner provided WM_G can simulate G and WM_H can simulate H.

Parallel Composition without Communication. Let $G = (\mathbf{G} = (G_0, G_1), X, Y, A, B, \partial_{0,G}, \partial_{1,G}, \gamma_{0,G}, \gamma_{1,G})$ and $H = (\mathbf{H} = (H_0, H_1), Z, W, C, D, \partial_{0,H}, \partial_{1,H}, \gamma_{0,H}, \gamma_{1,H})$ be CP automata. Then the noncommunicating parallel composition of G and H, written $G \times H$, is the CP automaton:

$$\begin{aligned} G \times H = (&\mathbf{G} \times \mathbf{H} = (G_0 \times H_0, G_1 \times H_1), X \times Z, Y \times W, A \times C, B \times D, \\ &\partial_{0,G \times H}(g, h) = \partial_{0,G}(g) \times \partial_{0,H}(h), \partial_{1,G \times H}(g, h) = \partial_{1,G}(g) \times \partial_{1,H}(h), \\ &\gamma_{0,G \times H} = \gamma_{0,G} \times \gamma_{0,H}, \ \gamma_{1,G \times H} = \gamma_{1,G} \times \gamma_{1,H}) \end{aligned}$$

This noncommunicating parallel composition still features synchronous communication, but this time of pairs of data values.

We model the noncommunicating parallel composition of G and H at the IWIM system level thus. Let WM_G and WM_H be IWIM systems representing G and H respectively. We assume that WM_G and WM_H have pure workers, pw_G and pw_H, one-state pure managers, pm_G and pm_H, external input channels $ch_{t,G}$ and $ch_{t,H}$, external output channels $ch_{s,G}$ and $ch_{s,H}$, that $\gamma_{0,G}(A)$ and $\gamma_{1,G}(B)$ can be identified with a set of states of pw_G, and that $\gamma_{0,H}(C)$ and $\gamma_{1,H}(D)$ can be identified with a set of states of pw_H. Then we proceed as follows to construct $WM_{G \times H}$.

There is the usual one-state pure manager $pm_{G \times H}$ as above. We build a corresponding pure worker $pw_{G \times H} = (\{p_i\}, \{p_0\}, (St_{G \times H}, Init_{G \times H}, Tr_{G \times H}))$ from pw_G and pw_H by defining $St_{G \times H} = St_G \times St_H$, $Init_{G \times H} = (Init_G, Init_H)$, and for $Tr_{G \times H}$, whenever we have a pair of transitions in Tr_G of the form s_G -p_i?ind_G-› $arc_{st,G}$ -p_0!$outd_G$-› t_G, and a pair of transitions in Tr_H of the form s_H -p_i?ind_H-› $arc_{st,H}$ -p_0!$outd_H$-› t_H, we form the $Tr_{G \times H}$ transition pair:

(s_G, s_H) -p_i?(ind_G, ind_H)-› $(arc_{st,G}, arc_{st,H})$ -p_0!$(outd_G, outd_H)$-› (t_G, t_H).

We can now identify $\gamma_{0,G \times H}(A \times C)$ with states corresponding to $\gamma_{0,G}(A) \times \gamma_{0,H}(C)$, and $\gamma_{1,G \times H}(B \times D)$ with states corresponding to $\gamma_{1,G}(B) \times \gamma_{1,H}(D)$; and the rest of the data for $WM_{G \times H}$ is routine.

It is obvious that $WM_{G \times H}$ is able to simulate $G \times H$ in a straightforward manner provided WM_G can simulate G and WM_H can simulate H.

Up to now, the in-conditions and out-conditions of the component CP automata have played a passive role; the next construction remedies this.

Restricted Sum. Let $\mathcal{G} = (\mathbf{G} = (G_0, G_1), X, Y, A, B, \partial_{0,G}, \partial_{1,G}, \gamma_{0,G}, \gamma_{1,G})$ and $\mathcal{H} = (\mathbf{H} = (H_0, H_1), X, Y, B, C, \partial_{0,H}, \partial_{1,H}, \gamma_{0,H}, \gamma_{1,H})$ be CP automata. Then the restricted sum of $\mathcal{G}$ and $\mathcal{H}$, written $\mathcal{G} + \mathcal{H}$, is the CP automaton:

$\mathcal{G} + \mathcal{H} = (\mathbf{G} + \mathbf{H} = (G_0 + H_0 / \sim_B$ where $\sim_B$ is the finest equivalence relation generated by $\gamma_{1,G}(b) \sim_B \gamma_{0,H}(b)$ (and we write $[g]_B$ for the equivalence class containing g), $G_1 + H_1)$,
$X, Y, A, C,$
$\partial_{0,G+H} = \partial_{0,G} + \partial_{0,H}, \partial_{1,G+H} = \partial_{1,G} + \partial_{1,H},$
$\gamma_{0,G+H} = \gamma_{0,G}\ ,\ \gamma_{1,G+H} = \gamma_{1,H})$

(As expected, the sources and targets of the arcs in $G_1 + H_1$ are the equivalence classes of the corresponding sources and targets in G_0 and H_0.)

Let WM_G and WM_H be IWIM systems representing $\mathcal{G}$ and $\mathcal{H}$ respectively. We assume that WM_G and WM_H have pure workers, pw_G and pw_H, one-state pure managers, pm_G and pm_H, external input channels $ch_{\mathrm{t},G}$ and $ch_{\mathrm{t},H}$, external output channels $ch_{\mathrm{s},G}$ and $ch_{\mathrm{s},H}$, that $\gamma_{0,G}(A)$ and $\gamma_{1,G}(B)$ can be identified with a set of states of pw_G via maps $\gamma_{w0,G} : A \rightarrow St_G$, $\gamma_{w1,G} : B \rightarrow St_G$, and that $\gamma_{0,H}(B)$ and $\gamma_{1,H}(C)$ can be identified with a set of states of pw_H via maps $\gamma_{w0,H} : B \rightarrow St_H$, $\gamma_{w1,H} : C \rightarrow St_H$. We proceed as follows to construct WM_{G+H}.

There is the usual one-state pure manager pm_{G+H} as above. We build a corresponding pure worker $pw_{G+H} = (\{p_{\mathrm{i}}\}, \{p_0\}, (St_{G+H}, Init_{G+H}, Tr_{G+H}))$ from pw_G and pw_H by defining:

$St_{G+H} = St_G + St_H / \sim_B$ where $\sim_B$ is the finest equivalence relation generated by $\gamma_{w1,G}(b) \sim_B \gamma_{w0,H}(b)$ (and we write $[s]_B$ for the equivalence class containing s)

$Init_{G+H} = [Init_G]_B$

$Tr_{G+H} = \{[s]_B \text{ -}p_{\mathrm{i}}?v\text{-}\rangle\ [t]_B \mid [s]_B, [t]_B \in St,\ s \text{ -}p_{\mathrm{i}}?v\text{-}\rangle\ t \in Tr_{G,\mathrm{I}} \cup Tr_{H,\mathrm{I}}\} \cup$
$\{[s]_B \text{ -}p_0!v\text{-}\rangle\ [t]_B \mid [s]_B, [t]_B \in St,\ s \text{ -}p_0!v\text{-}\rangle\ t \in Tr_{G,\mathrm{O}} \cup Tr_{H,\mathrm{O}}\}$

That this works as desired is conditional on the observation that in both pw_G and pw_H, the states picked out by $\gamma_{w0,G}, \gamma_{w1,G}, \gamma_{w0,H}, \gamma_{w1,H}$ are, so to speak, 'G_0-states' and not '*arc*-states'. This can be assured by choosing $\gamma_{w0,G}, \gamma_{w1,G}, \gamma_{w0,H}, \gamma_{w1,H}$ to be $\gamma_{0,G}, \gamma_{1,G}, \gamma_{0,H}, \gamma_{1,H}$ in the base case construction, whereupon it evidently persists through the binary combinator simulations we have described, and enables us to formally identify $\gamma_{0,G+H} = \gamma_{0,G}$ with a set of states of pw_{G+H} via $\gamma_{w0,G+H} : A \rightarrow St_{G+H} = \gamma_{w0,G} / \sim_B$ and to identify $\gamma_{1,G+H} = \gamma_{1,H}$ with a set of states of pw_{G+H} via $\gamma_{w1,G+H} : C \rightarrow St_{G+H} = \gamma_{w1,H} / \sim_B$. With this confirmed, the construction of $St_G + St_H / \sim_B$ results in a glueing of $s \text{ -}p_{\mathrm{i}}?ind\text{-}\rangle\ arc \text{ -}p_0!outd\text{-}\rangle\ t$ sequences only at their ends, and it then becomes easy to see that the given recipe gives us an IWIM system WM_{G+H} capable of simulating the CP automaton $\mathcal{G} + \mathcal{H}$, if WM_G simulates $\mathcal{G}$ and WM_H simulates $\mathcal{H}$.

Two points deserve comment. Firstly, [Katis et al. (2000)] speak of the need to 'adjust' the in-conditions or out-conditions of a CP automaton in order to make it fit for some particular purpose. More than anything else this is an indication that these interconnection aspects of the automaton are really properties that belong more to the interconnection mechanism itself, than to the automata involved.

Secondly if, following [Katis et al. (2000)], we intend the restricted sum to model sequential composition, the construction of WM_{G+H}, though faithful to the CP automaton $G+H$, suffers from the weakness that if a final state of G has out-transitions, and a corresponding initial state of H has in-transitions, then a run may wander from G to H and then back in to G. The IWIM system paradigm offers more flexibility here, allowing the expression of an irreversible transition from G to H. We describe the details, resulting in the construction of an IWIM system $WM*_{G+H}$ that simulates $G+H$ in a different way.

Suppose in $G_0 + H_0 / \sim_B$ above, there are k of the equivalence classes that are non-singletons, i.e. there are k classes that glue at least one element of G_0 to at least one element of H_0 (the remaining classes just containing individual elements outside the ranges of $\gamma_{1,G}(B)$ and $\gamma_{0,H}(B)$). Call them:

$$[\gamma_{w1,G}(b)_1], [\gamma_{w1,G}(b)_2] \ldots [\gamma_{w1,G}(b)_k]$$

Now partition each of $[\gamma_{w1,G}(b)_1] \ldots [\gamma_{w1,G}(b)_k]$ into two subsets each:

$$\begin{array}{lll} [\gamma_{w1,G}(b)_1]_G = [\gamma_{w1,G}(b)_1] \cap G_0 & \text{and} & [\gamma_{w1,G}(b)_1]_H = [\gamma_{w1,G}(b)_1] \cap H_0 \\ \quad\ldots \qquad\quad \ldots & & \quad\ldots \qquad\quad \ldots \\ [\gamma_{w1,G}(b)_k]_G = [\gamma_{w1,G}(b)_k] \cap G_0 & \text{and} & [\gamma_{w1,G}(b)_k]_H = [\gamma_{w1,G}(b)_k] \cap H_0 \end{array}$$

all nonempty by our assumptions. Replacing in St_{G+H} the $[\gamma_{w1,G}(b)_1] \ldots [\gamma_{w1,G}(b)_k]$ by the $[\gamma_{w1,G}(b)_1]_G, [\gamma_{w1,G}(b)_1]_H \ldots [\gamma_{w1,G}(b)_k]_G, [\gamma_{w1,G}(b)_k]_H$ is tantamount to generating a new equivalence relation, which we call $B*$, on the state space $St_G + St_H$. This is the finest relation generated by the two families of clauses:

$$(\gamma_{w1,G}(b) \sim_B \gamma_{w0,H}(b) = \gamma_{w0,H}(c) \sim_B \gamma_{w1,G}(c)) \Rightarrow \gamma_{w1,G}(b) \sim_{B*} \gamma_{w1,G}(c))$$

$$(\gamma_{w0,H}(b) \sim_B \gamma_{w1,G}(b) = \gamma_{w1,G}(c) \sim_B \gamma_{w0,H}(c)) \Rightarrow \gamma_{w0,H}(b) \sim_{B*} \gamma_{w0,H}(c))$$

Now we define:

$$\begin{array}{rl} St*_{G+H} = & (St_{G+H} - \{[\gamma_{w1,G}(b)_1] \ldots [\gamma_{w1,G}(b)_k]\}) \cup \\ & \quad \{[\gamma_{w1,G}(b)_1]_G, [\gamma_{w1,G}(b)_1]_H \ldots [\gamma_{w1,G}(b)_k]_G, [\gamma_{w1,G}(b)_k]_H\} \end{array}$$

$$Init*_{G+H} = [Init_G]_{B*}$$

$$\begin{array}{rl} Tr*_{G+H} = & \{[s]_{B*} \text{-}p_i?v\text{-}\!\!> [t]_{B*} \mid [s]_{B*}, [t]_{B*} \in St,\ s \text{-}p_i?v\text{-}\!\!> t \in Tr_{G,\mathrm{I}} \cup Tr_{H,\mathrm{I}}\} \cup \\ & \{[s]_{B*} \text{-}p_0!v\text{-}\!\!> [t]_{B*} \mid [s]_{B*}, [t]_{B*} \in St,\ s \text{-}p_0!v\text{-}\!\!> t \in Tr_{G,\mathrm{O}} \cup Tr_{H,\mathrm{O}}\} \cup \\ & \{[s]_{B*} \text{-}rec\text{-}\!\!> [t]_{B*} \mid s = \gamma_{w1,G}(b) = \gamma_{w0,H}(b) = t\ ,\ b \in B\} \end{array}$$

By distinguishing the G from the H components of the glueing states, we are able to introduce *rec* transitions from one to the other. All of these *rec* transitions are above the unique state of the pure manager, and all map to the identity reconfiguration on the corresponding port/channel network $(\{p_s, p_t\}, \{ch_s, ch_t\})$. Since the pure worker remains above this state when such a *rec* transition is executed, its *rec* transition completes and the run continues in the H component; however this time there is no way back to the G

component, even if there are in-transitions to the initial state of H used, and out-transitions from the final state of G reached.

This all works adequately, but is still open to the criticism that pure worker pw_G, its useful life over when the locus of control moves into the pw_H part of the system, remains alive, though defunct, preventing its resources from being reused. In a real system, it would be garbage collected releasing its resources for other activities. Equally, a demand driven implementation might well not create the pw_H part of the system until it was needed. Our IWIM system model enables us to express these aspects though we will not go into all the formal details. Here is the general idea.

We split the state of the pure manager into two; and (a modified) pw_G is above the new initial state, while pw_H is above the other state. There is a reconfiguration transition from the former to the latter, whose data is the identity reconfiguration on the port/channel network $(\{p_\mathrm{s}, p_\mathrm{t}\}, \{ch_\mathrm{s}, ch_\mathrm{t}\})$. The modification to pw_G entails adding the $[\gamma_{w1,G}(b)_1]_G \ldots [\gamma_{w1,G}(b)_k]_G$ states described previously to its state space, and then adding *rec* transitions to a typical $[\gamma_{w1,G}(b)_j]_G$ state from each of its comprising $\gamma_{w1,G}(b)_j$ states. These *rec* transitions map to the reconfiguration mentioned above.

It is clear that the behaviours of the resulting system are as follows. The manager starts in its initial state; consequently the modified pw_G is active. It executes until it reaches a $\gamma_{w1,G}(b)_j$ state and proceeds to perform the $\gamma_{w1,G}(b)_j$ -*rec*-› $[\gamma_{w1,G}(b)_j]_G$ transition. This maps to the reconfiguration step of the manager, and because pw_H is above the new manager state, the modified pw_G leaves the system configuration and pw_H joins it, starting in its initial state.

This story holds up if H has a unique initial state. If not, an unwinding technique similar to that used in our ABB system simulation must be employed.

Furthermore, the nontrivial state space now introduced for the manager has consequences for all the combinators. A product-like construction must be used on the manager states for the communicating and noncommunicating parallel compositions, while a sum-like construction, involving the introduction of reconfiguration transitions must be used for the restricted sum. We leave the fascinating details for the motivated reader.

7 Conclusions

In the preceding sections we have introduced a formal model for capturing some of the essence of the IWIM concept in an automata based framework. The essence of the IWIM model is the special role of reconfigurations, so our constructions aimed to reflect this in an explicit manner, rather than relying on 'programming them away' within a more general purpose automata theoretic framework. The objective was to model these structural aspects of reconfigurations involving managers and workers as simply as possible while keeping their special nature to the fore. This led to some complexity in the model as we saw, but not as much as there might have been had we chosen for example to model the full asynchrony of the true IWIM model, rather than emulate it via delay automata.

To keep things as accessible as possible, we started with elementary IWIM systems, before treating the unrestricted case. The fact that the generalisation went smoothly is due in no small way to the fact that the design of the model was tacitly undertaken in a manner in sympathy with categorical imperatives, which have a great capacity to foster relatively elegant structural properties.

Having built our IWIM systems and dealt with the emulation of full asynchrony, we emulated the ABB and KSW models. While the ABB model is a distributed state model like ours, the KSW model is a global state model, in which the state of the entire system resides at one indivisible point. This policy permits a straightforward construction for sequential composition, at the price of being somewhat unrealistic for a distributed model of computation. It is clear that sequential composition would involve the distributed termination problem in a distributed state model, and this is one reason why it is not contemplated for the ABB model. Had we not wanted to capture all the algebraic properties described in [Katis et al. (2000)], including sequential composition, we could have employed a more natural construction to emulate aspects of the KSW model. For example we could have piped the output of one worker into the input of another in modelling communicating parallel composition, this however immediately distributes the state.

Finally, we observe that coordination models different from the IWIM one, and in particular the global state tuple based approaches, must nevertheless embody the capacity for disentangling management from worker aspects, which was done so readily for IWIM, even if they only do so implicitly. The challenge of extracting this structure from so different looking starting points remains an intriguing issue to explore in future publications.

Acknowledgement

The work described in this paper was partially supported by the EU in the course of the KIT-INCO Project SEEDIS (Contract No. 962114).

References

[Agha (1986)] Agha G. (1986); Actors: A Model of Concurrent Computation in Distributed Systems. MIT Press.

[Arbab (1995)] Arbab F. (1995); Coordination of Massively Concurrent Activities. CWI Tech. Rep. CS-R9565.

[Arbab (1996)] Arbab F. (1996); The IWIM Model for Coordination of Concurrent Activities. *in*: Proc. COORD-96, Ciancarini, Hankin (eds.), LNCS **1061**, 34-56, Springer.

[Arbab et al. (1993)] Arbab F., Herman I., Spilling P. (1993); An overview of Manifold and its Implementation. Concurrency: Practice and Experience **5**, 23-70.

[Arbab et al. (1998)] Arbab F., Blom C. L., Burger F. J., Everaars C. T. H. (1998); Rusable Coordination Modules for Massively Concurrent Applications. Software: Practice and Experience **28**, 703-735.

[Arbab et al. (2000a)] Arbab F., de Boer F. S., Bonsangue M. M. (2000a); A Logical Interface Description Language for Components. *in*: Proc. COORD-00, Porto, Roman (eds.), LNCS **1906**, 249-266, Springer.

[Arbab et al. (2000b)] Arbab F., de Boer F. S., Bonsangue M. M. (2000b); A Coordination Language for Mobile Components. *in*: Proc. ACM SAC-00, 166-173.

[Banach et al. (2002)] Banach R., Arbab F., Papadopoulos G. A., Glauert J. R. W. (2002); A Multiply Fibred Automaton Semantics for IWIM. CWI Research Report SEN-R0206. `http://www.cwi.nl`

[Bonsangue et al. (2000)] Bonsangue M. M., Arbab F., de Bakker J. W., Rutten J. J. M. M., Scutellà A., Zavattaro G. (2000); A Transition System Semantics for the Control-Driven Coordination Language MANIFOLD. Theor. Comp. Sci. **240**, 3-47.

[Carriero and Gelernter (1989)] Carriero N., Gelernter D. (1989); LINDA in Context. Comm. ACM **32**, 444-458.

[Ciancarini and Hankin (1996)] Ciancarini P., Hankin C. H. L. (eds.) (1996); Coordination Languages and Models 1996 (Proc. COORD-96). LNCS **1061**, Springer.

[Ciancarini and Wolf (1999)] Ciancarini P., Wolf A. L. (eds.) (1999); Coordination Languages and Models 1999 (Proc. COORD-99). LNCS **1594**, Springer.

[Garlan and Le Metayer (1997)] Garlan D., Le Metayer D. (eds.) (1997); Coordination Languages and Models 1997 (Proc. COORD-97). LNCS **1282**, Springer.

[Gelernter (1985)] Gelernter D. (1985); Generative Communication in Linda. ACM Trans. Prog. Lang. Sys. **7**, 80-112.

[Katis et al. (2000)] Katis P., Sabadini N., Walters R. F. C. (2000); A Formalisation of the IWIM Model. *in*: Proc. COORD-00, Porto, Roman (eds.), LNCS **1906**, 267-283, Springer.

[Malone and Crowston (1994)] Malone T., Crowston K. (1994); The Interdisciplinary Study of Coordination. ACM Comp. Surv. **26**, 87-119.

[Omicini et al. (2002)] Omicini A., Zambonelli F., Klusch M., Tolksdorf R. (2002); Coordination of Internet Agents: Models, Technologies, and Applications. Springer.

[Papadopoulos and Arbab (1998)] Papadopoulos G. A., Arbab F. (1998); Coordination Models and Languages. *in*: Advances in Computers — The Engineering of Large Systems, Zelkowitz (ed.), 329-400, Academic.

[Porto and Roman (2000)] Porto A., Roman G-C. (eds.) (2000); Coordination Languages and Models 2000 (Proc. COORD-00). LNCS **1906**, Springer.

[Shapiro (1989)] Shapiro E. (1989); The Family of Concurrent Logic Languages. ACM Comp. Surv. **21**, 412-510.