URL: http://www.elsevier.nl/locate/entcs/volume66.html 15 pages

IWIM Semantics via Fibred Automata [1]

R. Banach [2]

Computer Science Department, Manchester University
Manchester, M13 9PL, U. K.

F. Arbab [3]

Software Engineering Department, CWI
Kruislaan 413, 1098 SJ Amsterdam, Netherlands

G. A. Papadopoulos [4]

Computer Science Department, University of Cyprus
75 Kallipoleos St., Nicosia, Cyprus

J. R. W. Glauert [5]

School of Information Systems, University of East Anglia
Norwich, NR4 7TJ, U.K.

Abstract

Coordination programming helps to separate concerns in the programming of the coordination activities in complex applications software. It separates the development, verification, maintenance, and reuse of the coordination and communication protocols, from the development of the rest of the application; coincidentally making these entities into standalone products. The IWIM coordination model is briefly reviewed, 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.

[1] Partially supported by the EU in the KIT-INCO Project SEEDIS (Contract No. 962114)
[2] Email: banach@cs.man.ac.uk
[3] Email: farhad@cwi.nl
[4] Email: george@cs.ucy.ac.cy
[5] Email: J.Glauert@sys.uea.ac.uk

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.

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 is subordinate. The somewhat greater flexibility of the latter tends to make it more popular in concurrent applications.

The observation that both models are really too low-level to serve as a foundation for the standalone development of protocols has led in recent years to an upsurge in activity in coordination frameworks and languages. An early survey is [17] which characterisies coordination as an emerging discipline. Various approaches with roots in eg. the actor model [1], or in logic programming [21], were instrumental in establishing coordination as an independent discipline. See [12,14,19,13,20,18] for representative contemporary work. A number of higher level perspectives have emerged. Among these are the tuple based approaches such as Linda [15,11], and by contrast, the connection control based approaches such as IWIM, the subject of this paper.

In the rest of this paper, we survey the IWIM model informally in Section 2. In Section 3 we develop an automaton-based model to express the essentials of IWIM, which we call the IWIM systems model. For lack of space, we restrict our attention to a special case of the model, the elementary IWIM

systems. 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 can shoulder the burden. We abstract away from the ability of workers to continue with internal actions on their own; thus our model falls short of capturing 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 [4,10].

In Section 4 we indicate briefly how our elementary IWIM systems model can be extended to model processes that are able to display both worker and manager traits, and we discuss how reconfigurations can be implemented asynchronously. We raise the issue of the algebraic combination of these IWIM systems and the contrast beween the general purpose approach to system combination, and the bespoke one arising from the emulation of the models of Arbab, de Boer and Bonsangue [6], and Katis, Sabadini and Walters [16]. The latter are two earlier theoretical models exploring aspects of IWIM. A fuller treatment of all these issues can be found in [8]. Section 5 concludes.

2 The IWIM Model

In this section we review the generic coordination framework known as the Ideal Worker Ideal Manager (IWIM) model [2,3,5]. 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 *man* 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 *man* 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.

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 our semantic model. The idea is to use a fibration-inspired strategy, to reflect the way that IWIM events tear down and rebuild interconnections between families of processes. Thus elementary IWIM automata will have in the base 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.

The capacity of IWIM systems to reconfigure themselves via events that provoke managers to perform 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,

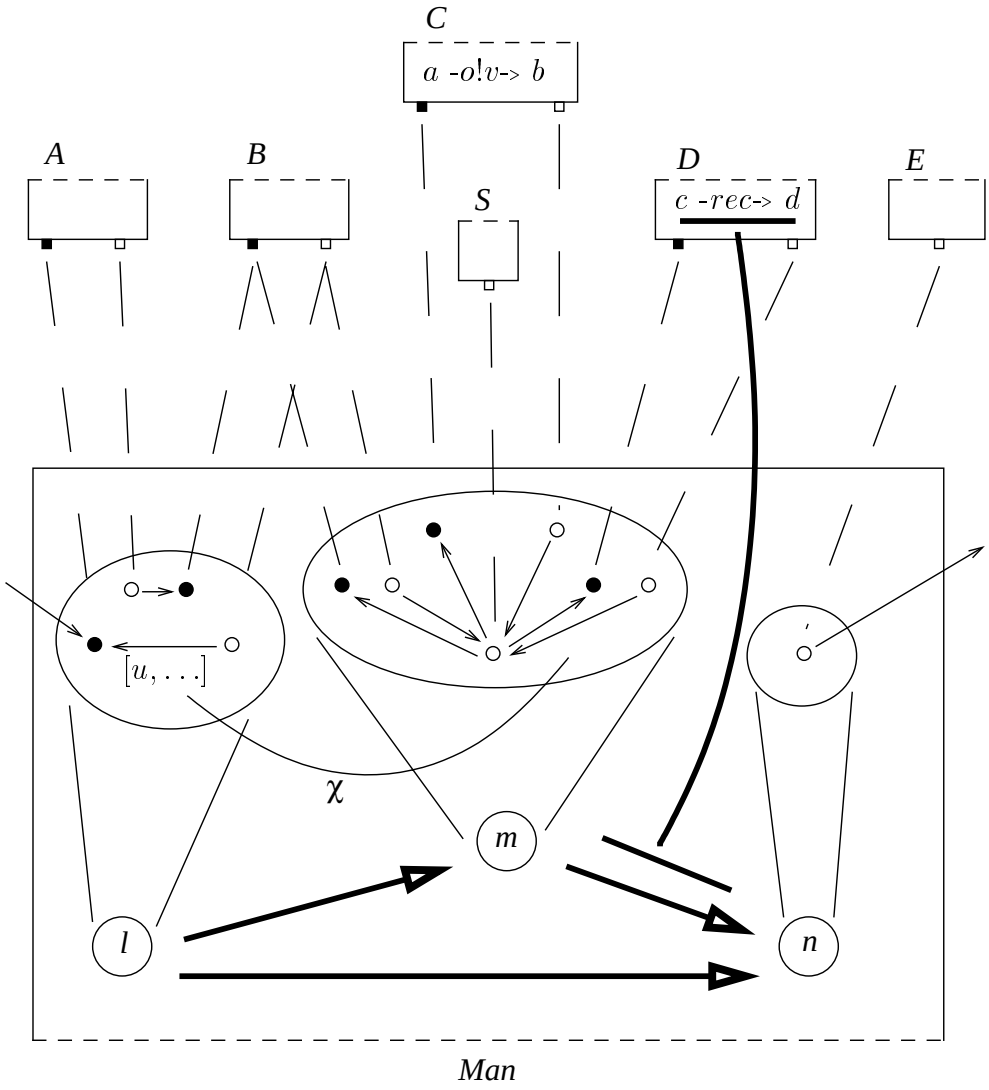

Fig. 1. An Elementary IWIM Automaton System.

resulting in reconfiguration). Unlike genuine IWIM systems, this is a synchronous activity in our model, but we will indicate 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 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 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 provoke reconfigurations implemented by the manager.

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 $s_m, t_m : C_m \to P_m$ that send channels to source and target port names where they 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 $\mathrm{dom}(s_m)$ are called external input channels, and channels not in $\mathrm{dom}(t_m)$ are called external output channels; channels in both $\mathrm{dom}(s_m)$ and $\mathrm{dom}(t_m)$ are called internal channels. In a reconfiguration transition, written m -r-> n, the r is shorthand for a partial injection on the channel names $\chi_{m,n} : C_m \to C_n$. 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, disjoint from O a set of output ports; and $A = (St, Init, Tr)$ is an automaton with states St, of which $Init \in St$ is an initial state, and $Tr \subset St \times Act \times St$ 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 $Tr_{\mathrm{I}} = \{a$ -$in?v$-> $b \in Tr\}$, $Tr_{\mathrm{O}} = \{a$ -$out!v$-> $b \in Tr\}$, $Tr_{\mathrm{R}} = \{a$ -rec-> $b \in Tr\}$, so that $Tr = Tr_{\mathrm{I}} \cup Tr_{\mathrm{O}} \cup Tr_{\mathrm{R}}$, the union being evidently disjoint. Additionally we define $Rec = \{rec \mid a$ -rec-> $b \in Tr\}$, 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) comprises 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:

(i) There is a relation $\hat{}$ between Wor and the management states of Man. We write $w\hat{}m$ to say that a worker w is above a management state m if the pair is in the relation.

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

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

(iv) Associated to each channel $c \in C_m$ (where m is a management state), there is a queue of messages which we write $c{:}[u_0, u_1, \ldots]$. 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:

(i) a state m of Man;

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

(iii) a set $qs = \{c : q_c \mid c : q_c = c : [u_0, u_1, \ldots], c \in C_n, n \in M\}$ of queues of messages $c{:}[u_0, u_1, \ldots]$ one per channel per 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).

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

(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).

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

(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.

$$\begin{array}{l} k\text{\^{}}m \ , \ a_k \in ests \ , a_k \text{ -}i?u\text{-> } b_k \ , \\ \lambda_m(p) = i \in I_{wor(k)} \ , \\ t_m(c) = p \ , \\ ests_{\mathrm{rest}} = ests - \{a_k\} \ , \\ qs_{\mathrm{rest}} = qs - \{c{:}[u, u_1, \ldots]\} \\ \hline m \longrightarrow m \ , \\ ests \longrightarrow ests_{\mathrm{rest}} \cup \{b_k\} \ , \\ qs \longrightarrow qs_{\mathrm{rest}} \cup \{c{:}[u_1, \ldots]\} \end{array}$$

(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.

$$\frac{\begin{array}{l} k\hat{}m\ ,\ a_k \in ests\ , a_k\ \text{-}o!u\text{->}\ b_k\ , \\ \lambda_m(p) = o \in O_{wor(k)}\ , \\ \emptyset \neq Out = \{d \mid s_m(d) = p\}\ , \\ ests_{\text{rest}} = ests - \{a_k\}\ , \\ qs_{\text{rest}} = qs - \{d{:}[\ldots, u_{d,n_d}] \mid d \in Out\} \end{array}}{\begin{array}{l} m \longrightarrow m\ , \\ ests \longrightarrow ests_{\text{rest}} \cup \{b_k\}\ , \\ qs \longrightarrow qs_{\text{rest}} \cup \{d{:}[\ldots, u_{d,n_d}, u] \mid d \in Out\} \end{array}}$$

(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.

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

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 $r_{k_r\hat{}m}$. 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.

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

$m \longrightarrow n$,
$ests \longrightarrow ests_{\text{rest}} \cup \{b_{k_r}\}$,
$qs \longrightarrow qs_{\text{rest}} \cup qs_{\text{dom}} \cup qs_{\text{merge}}$

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.

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 occurring in $EConfs(Man, Wor)$. These are given by a function $\mathrm{e}\pi_{\mathrm{man}}$ where $\mathrm{e}\pi_{\mathrm{man}}(m, ests, qs) = m$. 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 $\mathrm{e}\pi_{\mathrm{man}}(m, ests, qs) \longrightarrow \mathrm{e}\pi_{\mathrm{man}}(m', ests', qs')$, 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. Consider* $\mathrm{EConfs}(Man, Wor)$*, the associated transition system, and $Mngr(Man, Wor)$, the corresponding set of manager transitions. Then there is a projection:*

$$\Pi_{\mathrm{e}} : \mathrm{EConfs}(Man, Wor) \to Mngr(Man, Wor)$$

which maps states by:

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

and which maps (REC) transitions by:

$$\begin{array}{ccc} (m, ests, qs) & \longrightarrow & (m', ests', qs') \\ \quad\quad\mapsto & & \quad\quad\mapsto \\ m \longrightarrow m' & = & \mathrm{e}\pi_{\mathrm{man}}(m, ests, qs) \longrightarrow \mathrm{e}\pi_{\mathrm{man}}(m', ests', qs') \end{array}$$

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

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

Proof. Obvious. □

4 Properties and Extensions of IWIM Systems

In this section we outline some aspects of our IWIM systems model that lack of space prevents us from treating in a more comprehensive manner. The first issue concerns the fact that in the general case, processes in IWIM are capable of

displaying both manager and worker behaviour. To address this, a more comprehensive formal construction asynchronously combines a worker automaton and a manager automaton of the kind we have seen above in elementary IWIM systems, to yield a worker-manager automaton. The asynchronous product construction has a state space which is the cartesian product of the individual state spaces, permitting moves which are either worker or manager moves, each acting on their respective component. The relatively decoupled nature of the construction means that all the apparatus for linking workers to their managers carries over without alteration from the elementary case. Consequently workers may be simultaneously managed by several different managers, just as a manager can control several workers. The only technical point of note, is that reconfiguration transitions are now synchronised across as many managers as the poor worker is currently controlled by. Since this synchronous reconfiguration aspect of the model is at odds with the true nature of IWIM, we emphasise that we can recover asynchronous reconfiguration by simulation, this being the second issue on our list.

Asynchronous reconfiguration is in fact relatively easy to simulate by the introduction of delay automata; one such automaton for every occurrence of a worker being above a manager state. The purpose of the delay automata is to buffer the events in transit from the original worker to the original manager. Thus instead of an original worker raising an event as previously, it posts a message encoding the event required on a special purpose port, which is broadcast over channel connections to the delay automata corresponding to all the managers the worker is above. These automata then provoke the necessary reconfigurations one at a time at their leisure.

A third issue concerns the possibilities for combining IWIM automata in various ways. Since IWIM automata contain a variety of attributes, a large number of possibilities arise. These fall broadly into two classes, the general purpose ones and more specific constructions. Among the former are a number of fairly natural pushout and pullback constructions which exist under relatively straightforward conditions. These have to be built up stage by stage to deal with all the layers of detail precisely enough.

Among the latter, are constructions which only apply to automata of some precise form, which arises because the automata are designed to be emulations of some other system. Pertinent cases in point come from the Arbab, de Boer and Bonsangue model [6], and the Katis, Sabadini and Walters model [16]. These are two formal models for exploring theoretically some of the features of IWIM. Generic constructions for IWIM automata (in the sense of this paper) can be given, that accurately reflect the workings of these models. Moreover where these models possess algebraic combinators of their own (cf. the Katis, Sabadini and Walters model in particular), the natural way of combining the emulations turns out not to arise from the generic combinators for IWIM automata, but from ad hoc constructions valid only because of the strong invariants possessed by these emulations. Full details can be found in [8].

5 Conclusions

In the preceding sections we have reviewed the essentials of the IWIM model, and constructed elementary IWIM systems as automata families that capture some of the characteristics of IWIM, or of concrete implementations of IWIM such as MANIFOLD, in an abstract way. We have concentrated on the elementary IWIM systems because they illustrate the most important features of this method of modelling IWIM-style coordination in a fibration-oriented scenario. We went on to discuss in the last section, a number of issues that lack of space prevented us from giving a full treatment to. Aside from these issues that we touched upon, there remains the interesting question of how one might retrieve similar structures to the present ones in the shared store models, which must after all be capable of exhibiting the same range of behaviours as the IWIM model.

References

[1] Agha G. "Actors: A Model of Concurrent Computation in Distributed Systems," MIT Press, 1986.

[2] Arbab F. "Coordination of Massively Concurrent Activities." CWI Tech. Rep. CS-R9565, 1995.

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

[4] Arbab F., Herman I., Spilling P. "An overview of Manifold and its Implementation." Concurrency: Practice and Experience **5**, 23-70, 1993.

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

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

[7] Arbab F., de Boer F. S., Bonsangue M. M. "A Coordination Language for Mobile Components." *in*: Proc. ACM SAC-00, 166-173, 2000.

[8] Banach R., Arbab F., Papadopoulos G. A., Glauert J. R. W. "A Multiply Fibred Automaton Semantics for IWIM." *submitted*, 2002.

[9] Best E., Devillers R., Koutny M. "Petri Net Algebra." Springer, 2000.

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

[11] Carriero N., Gelernter D. "LINDA in Context." Comm. ACM **32**, 444-458, 1989.

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

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

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

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

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

[17] Malone T., Crowston K. "The Interdisciplinary Study of Coordination." ACM Comp. Surv. **26**, 87-119, 1994.

[18] Omicini A., Zambonelli F., Klusch M., Tolksdorf R. "Coordination of Internet Agents: Models, Technologies, and Applications." Springer, 2002.

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

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

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