Control-Based Coordination of Human and Other Activities in Cooperative Information Systems

George A. Papadopoulos

Department of Computer Science
University of Cyprus
75 Kallipoleos Str, P.O.B. 537
CY-1678 Nicosia, Cyprus

E-mail: george@turing.cs.ucy.ac.cy

Farhad Arbab

Department of Interactive Systems
CWI
Kruislaan 413, 1098 SJ Amsterdam
The Netherlands

E-mail: farhad@cwi.nl

1 Introduction

Modelling of activities within an information system or between different information systems has become a complex task. Performing these activities (often known as *groupware, workflow, electronic commerce* and *enterprize reengineering*) is often done in conjunction with computer-based cooperative environments such as electronic mail, voice and video teleconferencing, electronic classrooms, etc. Modelling these activities has become a task which often is not possible to perform by single persons, but by groups of people, often even distributed over different organisations, countries, etc.

Recently, we have seen the use of coordination languages to model activities in information systems, and, in particular, variants of the so called Shared Dataspace Model, its most prominent member being Linda ([1]). Although Linda is indeed a successful coordination model, when it is evaluated from the point of view of acting as a framework for modelling human and other activities in information systems, it has some serious deficiencies which carry over to all the other related models that are based on it. These deficiencies are: (i) It is data-driven which may not be very natural since we are interested more in how the *flow of information* between the involved agents is set-up and how an agent *reacts* to receiving some information, rather than *what kind of data* it sends or receives. (ii) The shared dataspace as a metaphor may not be very intuitive since the sharing it encourages and imposes contrasts with how information actually flows within an organisation; people do not take the work to be done by others to common rooms where from other people pass by and pick the work up. (iii) Furthermore, the shared dataspace is not secure and it is possible for information posted there to be lost or forged.

In the next section we present a different approach for modelling such activities where: (i) communication between agents is done by means of point-to-point *stream* connections; (ii) the agents comprising a coordination pattern are defined by means of being in one of a number of predefined *states,* where a state is a set of observable

stream connections; (iii) evolution of a community of coordinators is *event-driven* (or control based) in the sense that the agents in question observe the presence of events and react accordingly.

2 An Event-Driven Control-Based Modelling Framework

In this section we describe a framework for modelling activities in organisations based on MANIFOLD and its underlying coordination model IWIM ([2]). Our framework is essentially a three level one: (i) the top part is an easy to use visual interface which defines graphically the interrelationships and behaviour of the involved agents; (ii) the middle part is a verbal (semi-formal) description of the states defining each agent and how it reacts to receiving some event; (iii) the lower part is the actual implementation of the scenario to be modelled in MANIFOLD.

In our model, all entities participating in an activity (humans, devices, CSCW tools, shared resources such as active or passive documents, etc.) are *agents*. We distinguish two categories of agents: *worker* agents which perform computational work (whatever that may be according to the specific details of some particular scenario) and *manager* agents which are responsible for coordinating the activities of worker agents. Note that manager agents may themselves be seen as worker agents from other, higher up in the hierarchy, manager agents. However, the genuine worker agents (i.e. those performing some actual computational task) such as computer programs, CSCW tools, hardware devices, etc. form the bottom level of the layer and cannot be subdivided any further. We are not concerned with the internal details of these agents, only with their interaction with their environment.

Every agent, whether it is a manager or a worker agent, communicates with its environment by means of at least one *input port* and one *output port.* In general, an agent may have more than one input and/or output port. Furthermore, every agent observes a number of *events* and reacts to them accordingly. However, it should be made clear that the purpose of an event is to make one or more agents aware of some situation that must be handled and not to transfer actual data — this is the purpose of *streams*. More to the point, agents communicate actual data between themselves by means of connecting respective pairs of their input and output ports via *streams.* Finally, every agent is defined at any moment in time by means of being in some *state.* It is known beforehand which states an agent can be in during the lifespan of its activities. In order for some agent to change its state, it must observe the raising of a particular event. Reacting to an event (and therefore changing the current state) typically means establishing new stream connections between input and output ports and abandoning old connections.

The above general description of our model has some clear advantages over the traditional Linda-like approaches we have seen so far: (i) Every worker agent is only concerned with getting workload from its input port(s), performing the required work for which it is responsible, and putting the outcome to its output port(s). Its only other

communication with its environment is by means of observing (if at all) any events. (ii) Every manager agent is only concerned with making sure that the output produced by some worker agents are sent to some other worker agents that require it. The manager identifies workers by means of their responsibilities and workflow interdependencies, not the actual work (data) they produce. (iii) The model is inherently secured and flexible (new agents can come and go dynamically).

As an example we model a scenario where (mainly) four agents are involved, as it happens all of them being humans, collaborating in the development of some document. The first agent is the *author* who is responsible for writing up the document as well as performing significant changes to it. The second agent is the *editor* who checks the document's validity, performs any corrections and, if necessary, returns the document back to the author for further substantial changes. The third agent is the *manager* (still a worker process though, as far as our model is concerned) which either approves the document or sends it back to the editor. The fourth agent is a true coordinator agent responsible for managing the workflow between the other three agents (it could represent a department's supervisor). We also assume the presence of other agents (especially atomic agents performing purely computational work). In the sequel and for reasons of brevity we present the code for levels ii and iii only and for three of the agents involved, namely author, editor and manager. Level ii is described in terms of the states an agent can be in and what event it must observe to make a transition to another state; level iii is the MANIFOLD code.

Author

State 0: Receive document in in-port. State 1: Produce or modify document. State 1a: Put document in out-port. State 2: Recur from the beginning. State 3: Request to be substituted.

```
manifold Author (event i_had_enough)
{
 event prod_amend_doc, doc_ready, send_doc, time_to_go, flushed.

 begin:(guard(input,full,prod_amend_doc),          // State S0
        terminated(void)).
 prod_amend_doc:{process writer is Word_Program(doc_ready).
                                                   // State S1
                 begin:(activate(writer),
                        input->writer,
                        terminated(void)).
                 doc_ready:(writer->output,        // State S1a
                            post(send_doc)).
                }.
 send_doc:(post(begin)).                           // State S2
 time_to_go:{begin:(raise(i_had_enough),           // State S3
                    guard(output,a_disconnected,flushed),
                    terminated(void)).
             flushed:halt.
            }
}
```

Editor

State 0: Receive document in in-ports. State 1: Check document. State 1a: Send document to the manager. State 1b: Send document back to the author. State 2: Forward document back to the author.

```
manifold Editor (port in from_author, from_man, port out to_author,
to_man)
{
 event check_doc, doc_ok, doc_not_ok, send_doc_back, send_doc_man.

 begin:(guard(from_author,full,check_doc),          // State S0
        guard(from_man,full,send_doc_back),
        terminated(void)).
 check_doc:{process checker is Speller(doc_ok, doc_not_ok).
                                                    // State S1
            begin:(activate(checker),
                   from_author->checker,
                   terminated(void)).
            doc_ok:(checker->to_man,                // State S1a
                    guard(checker.output,empty,begin),
                    terminated(void)).
            doc_not_ok:(checker->to_author,         // State S1b
                        guard(checker.output,empty,begin),
                        terminated(void)).
           }
 send_doc_back:(from_man->to_author,                // State S2
                post(begin)).
}
```

Manager

State 0: Receive document in in-port. State 1: Verify and forward document. State S2: Ask for more work.

```
manifold Manager (event more_work, port out to_dept_head, to_editor)
{
 event verify_doc.

 begin:(guard(input,full,verify_doc),               // State S0
        terminated(void)).
 verify_doc:(if (document is ok)                    // State S1
             then input->to_dept_head
             else input->to_editor,
             guard(input,empty,begin).
 … raise(more_work).                                // State S2
}
```

References

[1] S. Ahuja, N. Carriero and D. Gelernter, "Linda and Friends", *IEEE Computer* **19(8)**, Aug. 1986, pp. 26-34.

[2] F. Arbab, "The IWIM Model for Coordination of Concurent Activities", *1st International Conference on Coordination Models, Languages and Applications (Coordination'96)*, Cesena, Italy, 15-17 April, 1996, LNCS 1061, Springer Verlag, pp. 34-56.