SCI 2000 - 4th World Muticonference on Systemics, Cybernetics and Informatics Proceedings, Vol.1. p.195 - 200. Orlando, USA. International Institute of Informatics and Sytemics (IIIS), 2000.

Coordinating Activities in Collaborative Environments: A High Level Petri Nets Based Approach

Alberto B. Raposo, **Léo P. Magalhães** and **Ivan L. M. Ricarte**
State University of Campinas (UNICAMP)
School of Electrical and Computer Engineering (FEEC)
Department of Computer Engineering and Industrial Automation (DCA)
CP 6101 – 13083-970 – Campinas, SP, Brazil
{alberto, leopini, ricarte}@dca.fee.unicamp.br

ABSTRACT

The coordination of interdependencies among activities in collaborative environments is a very important and difficult task. In this paper we present a set of coordination mechanisms for the specification and control of interaction among collaborative activities. To model these mechanisms, we use high level Petri nets, which have proven to be an adequate approach to evaluate the behavior of a computer supported collaborative system before its implementation.

Keywords: Coordination, Collaborative Environments, Petri Nets, Computer Supported Cooperative Work, Multiuser Interaction.

1. INTRODUCTION

Some activities involving multiple individuals do not require a formal planning. Activities associated to the social relations are generally well coordinated by the "social protocol", which is characterized by the absence of any coordination mechanism among activities, trusting the participants' abilities to mediate interactions. Examples of computer supported activities of this kind are the chats and videoconferences.

On the other hand, activities related to cooperative *work* (not social relations) require sophisticated coordination mechanisms to avoid that participants get involved in conflicting or repetitive tasks.

This paper focuses on the coordination of activities in computer supported collaborative environments, defining a set of interdependencies that frequently occur among collaborative tasks and presenting coordination mechanisms for these dependencies. The idea is to separate activities (tasks) from dependencies (controlled by the coordination mechanisms), enabling the use of different coordination policies in the same collaborative environment, by changing only the coordination mechanisms. Moreover, the coordination mechanisms can be reused in other collaborative environments.

This paper is organized as follows. In the next section we introduce high level Petri nets, which is the tool we use to model the coordination mechanisms. Then, in Section 3, we make a brief overview of works related to coordination in Computer Supported Cooperative Work. In Section 4 the dependencies and coordination mechanisms are presented, and in Section 5 an example of use of the mechanisms is shown. The conclusions and future work are discussed in Section 6.

2. HIGH LEVEL PETRI NETS

Petri nets (PNs) are a modeling tool applicable to a variety of fields and systems, specially suitable for systems with concurrent events. Murata [11] presents a very good introduction to the theme. Formally, a PN is defined as a 5-tuple (P, T, F, w, M_0), where: $P = \{P_1, ..., P_m\}$ is a finite set of places; $T = \{t_1, ..., t_n\}$ is a finite set of transitions; $F \subseteq (P \times T) \cup (P \times T)$ is a set of arcs; $w: F \rightarrow \{1, 2, ...\}$ is a weight function; $M_0: P \rightarrow \{0, 1, 2, ...\}$ is the initial marking; with $(P \cap T) = \varnothing$ and $(P \cup T) \neq \varnothing$.

In a PN model, states are associated to places and tokens, and events to transitions. A transition t is said to be enabled if each input place $P_i \in \bullet t$ is marked with at least $w(P_i, t)$ tokens, where $w(P_i, t)$ is the weight of the arc between P_i and t. Once enabled, a transition will fire when its associated event occurs. Firing transition t, $w(P_i, t)$ tokens are removed from each input place P_i and $w(t, P_o)$ tokens are added to each output place $P_o \in t\bullet$. Here, $\bullet t$ and $t\bullet$ means, respectively, the set of input and output places of transition t.

A useful notation for PNs is the graphical notation, where circles represent places, rectangles represent transitions, dots represent tokens and arrows represent the arcs, with weights above. By definition, an unlabeled arc has weight 1.

In addition to the basic PN model, several extensions appears in the literature. In this paper we use three of them: inhibitor arcs, nets with time and high level nets.

An inhibitor arc connects a place *P* with a transition *t* and enables *t* only if *P* has no tokens. In the graphical notation, inhibitor arcs are represented with a circle on the edge. One way to include the notion of time in PNs is to associate it with transitions firing. In this case, tokens are removed from input places of a transition and some time later (firing time) are added to the output places. This kind of non-instantaneous firing is called firing with token reservation.

High level PNs include, among other types, predicate/transition nets and colored PNs [4]. The most important characteristic of a high level PN is the distinction of tokens (called colored tokens). The arcs have labels defining variables (or constants) that dictate how many and which kind of tokens will be removed from or added to the places. The same variable appearing in the incoming and outgoing arcs of a transition denotes the same token type. A transition is enabled if there is at least one possibility of consistent substitution of variables into typed tokens. In the example of Figure 1, transition *t1* is enabled because there are two tokens of type *a* in *P1*, being possible to substitute variable *<x>* for type *a*. After the firing of *t1*, *P1* remains with token *b* and *P2* receives a token of type *a*.

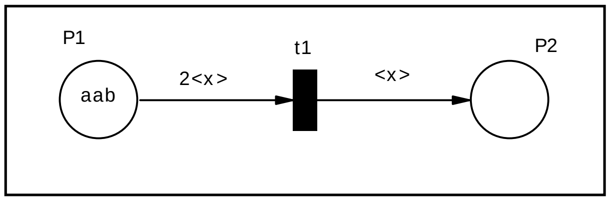

Figure 1: Example of a high level PN.

Besides their modeling capabilities, PNs have also a strong theoretical support for analysis and a number of simulation techniques. There are three kinds of analysis applicable to a PN model; verification, validation and performance analysis [2]. The verification analysis is used to guarantee that the net is correctly defined. It is verified whether the net has deadlocks, whether there are dead transitions, whether it reaches any undesired state, among others. In the validation analysis, it is checked whether the model works as expected. Tests are made via iterative simulations of fictitious cases to ensure that the model treats them correctly. Finally, the performance analysis evaluates the capacity of the system to achieve requisites such as average waiting time, throughput times, resource use, and so on.

3. RELATED WORK

In spite of their recognized importance, coordination mechanisms have not been included in the first collaborative systems. Only in the second half of the 80's the first systems with some kind of coordination mechanism were developed. One of the most representative system of that generation was the Coordinator [13], which has been developed based on linguistic theories and whose goal was to help email communication. Since that time, there has been examples of the use of PNs to coordinate activities in collaborative environments [6], [9].

In the 90's, systems have been constructed with more flexible and accessible coordination mechanisms. Inspired by coordination languages [8], which proposed the separation of computation from coordination for multi-threaded applications, some collaborative systems separates the implementation of coordination components from their other parts. This allows more flexibility in the use of coordination policies. COCA (Collaborative Objects Coordination Architecture) [10] and Trellis [7] are examples of systems of this kind. The Trellis, in particular, uses a variation of PNs in a server to specify interaction protocols for a group of collaborative clients.

Our work is more generic than those presented above. We define a set of interdependencies among tasks and associated coordination mechanisms (modeled by high level PNs) that can be used in workflows, multiuser interaction, virtual environments, etc. The main goal is not to implement a closed system, but to provide a set of mechanisms to enable the collaborative system designer to preview and test its behavior before implementing it, detecting possible problems.

4. COORDINATION MODELS

This work intends to provide mechanisms to manage interdependencies among collaborative tasks and guarantee that these dependencies will not be violated. The idea is that the designer of a collaborative environment be concerned only with the definition of tasks and their interdependencies, and not with the management of those dependencies.

In the proposed schema, an environment is modeled in two distinct levels, *workflow* and *coordination*. In the *workflow* level, the tasks and their interdependencies are defined. In the *coordination* level, interdependent tasks are expanded and the adequate coordination mechanisms are inserted among them.

During the passage from the workflow to the coordination level, each task which has an interdependency with another is expanded according to the model of Figure 2 [1]. The five places associated to the expanded tasks represent the interface with the resource manager and the agent that executes the tasks. Place *request_resources* indicates to the resource manager that the task needs a resource. After the assignment of the resource, the manager puts a token in place *assigned_resources* to continue the task. Places *start_tasks* and *finish_tasks* indicates, respectively, the beginning and the end of the task. Finally, *release_resources* indicates to the resource manager that the task has finished and the resource is available again.

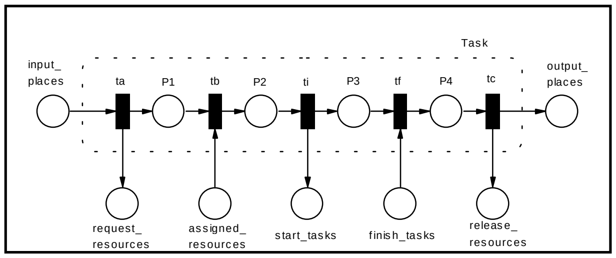

Figure 2: An expanded task in the coordination level.

The main goal of our work is to construct the coordination level from the workflow level, because once the interdependencies are defined, the expansion of tasks according to the model of Figure 2 and the insertion of coordination mechanisms can be automated.

The interdependencies presented below are divided into two classes: temporal and resource management.

Temporal Dependencies

Temporal dependencies establish the execution order of tasks. Coordination mechanisms associated to this kind of dependency have *start_tasks* as input place and *finish_tasks* as output place (Figure 2).

The proposed mechanisms are based on temporal relations defined in a classic paper of temporal logic [3]. In this paper, a set of primitive and mutually exclusive relations between time intervals is defined. We adapted these relations for the definition of temporal dependencies between tasks in collaborative environments, adding a couple of new relations and a few variations of those originally proposed. The following temporal dependencies are defined.

- *task 1 equals task 2*: both tasks must be executed simultaneously.
- *task 1 starts task 2*: Allen's original definition establishes that both tasks must start together and task 1 must finish before task 2 (we called this relation *startsA*). We also relaxed the second part of the definition, creating a variation of the relation in which it does not matter which task finishes before (*startsB*).
- *task 1 finishes task 2*: the original definition establishes that both tasks must finish together, but task 1 has to start after task2 (*finishesA*). As in the previous case, we created a new dependency relaxing the restriction on which task must start before (*finishesB*).
- *task 2 after task 1*: task 2 may only be executed after the execution of task 1. Two variations are possible for this relation. In the first one (*afterA*) each execution of task 1 enables a single execution of task 2. In the second variation, several executions of task 2 are enabled after a single execution of task 1.
- *task 1 before task 2*: from the temporal logic point of view, this dependency can be seen as the opposite of the previous one, but it generates a totally different coordination mechanism. Essentially, the difference is because in this case, the restriction occurs in the execution of task 1, which may not be executed anymore if task 2 has already been executed. Here, task 2 does not wait for the execution of task 1, which was the case for *task 2 after task 1*.
- *task 1 meets task 2*: task 2 starts immediately after the end of task 1.
- *task 1 overlaps task 2*: the original definition establishes that task 2 must start before the end of task 1, which must finish before task 2 (*overlapsA*). Relaxing the restriction that task 1 must finish first, we defined a variation of this relation (*overlapsB*).
- *task 2 during task 1*: two variations are possible. In the first one (*duringA*), task 2 can be executed only once during the execution of task 1. In the second one (*duringB*), task 2 can be executed several times.

Since the goal of the coordination mechanisms is to deal with relations that sometimes belongs to complex procedures, it is interesting to add mechanisms to avoid frequent deadlocks. One of such mechanisms is the use of timeouts. We defined two kinds of timeouts. In the first one (*timeoutA*), an alternative task is defined if the original one does not start after a certain waiting time. The other kind of timeout (*timeoutB*) returns the tokens to the input places of the task after a waiting time, enabling the following of alternative paths that do not execute the blocked task.

Resource Management Dependencies

Coordination mechanisms for resource management are complementary to those presented in the previous section and can be used in parallel to them. This kind of coordination mechanism deals with the distribution of resources among the tasks, and have *request_resources* and *release_resources* as input places and *assigned_ resources* as output place (Figure 2). We define three basic mechanisms for resource management:

- *Sharing*: a limited number of resources needs to be shared among several tasks.
- *Simultaneity*: a resource is available only if a certain number of tasks requests it simultaneously.
- *Volatility*: indicates whether, after the use, the resource is available again.

We have also defined composite mechanisms from the basic ones discussed above. For example, *sharing M* + *volatility N* indicates that up to *M* tasks may share the resource, which can be used *N* times only.

Differently from temporal dependencies, resource management dependencies are not binary relations. It is possible, for instance, that more than two tasks share a resource. Moreover, each of the above mechanisms requires a parameter indicating the number of resources to be shared, the number of tasks that must request a resource simultaneously, or the number of times a resource can be used (volatility).

A Language for the Definition of Interdependencies

In order to define the interdependencies among tasks, we have created a language that describes, one at each line, all the dependencies of a collaborative environment. The interdependencies are defined according to the following syntax `<dependency name> [parameters] “<task1 name>” “<task2 name>” [“<taskn name>”] [timeouts]`, where parameters are needed for resource management dependencies and the list of timeouts is optional.

Modeling Coordination Mechanisms using High Level Petri Nets

Initially, we have modeled coordination mechanisms for all dependencies discussed above using ordinary PNs [12]. However, a typical problem in the use of ordinary PNs is the state explosion, which can occur in our context when the number of interdependencies increases. High level PNs reduce this problem because they generate simpler models, with less places and transitions. Therefore, we remodeled them using high level PNs.

To illustrate one of the coordination mechanisms, Figure 3 shows the mechanism for simultaneity 2 (a resource is available only if two tasks request it simultaneously). In the figure, the arcs with expression *<x>+<y>* ensure that two different tasks (tokens of different colors) which are requesting the resources (tokens *r*) are going to receive them if they are available in place *Pn*.

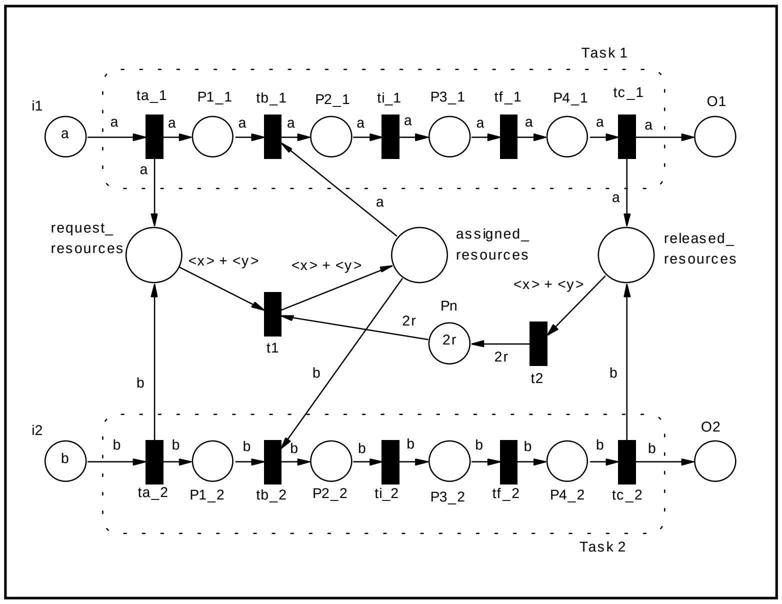

Figure 3: Coordination mechanism for *simultaneity 2.*

5. EXAMPLE

To illustrate the use of the coordination mechanisms, we present an example describing a typical situation in multiuser interaction. In our hypothetical environment, two users interact by means of a whiteboard. The workflow level of this environment is shown in Figure 4. As can be seen in the model, each user can “enter” the whiteboard, then write on it several times before “leaving” it. The following interdependencies appears in the model. The whiteboard is available only if both uses request it simultaneously (simultaneity 2); only one user may write on the whiteboard at each time (sharing a resource, e.g., a pen); and when user A leaves the whiteboard, user B must follow him/her (meets).

Using the language for the definition of interdependencies, the following file is written for the example above:

```
sim 2 “enterA” “enterB” time_outB
                              time_outB
div 1 “writeA” “writeB” time_outB
                              time_outB
meets “leaveA” “leaveB”
```

The model of the coordination level for this example is shown in Figure 5. At first glance, this model may seem complicate, but it is highly modular and easily built from the model of Figure 4, by expanding interdependent tasks (open rectangles in Figure 4) and inserting the predefined coordination mechanisms.

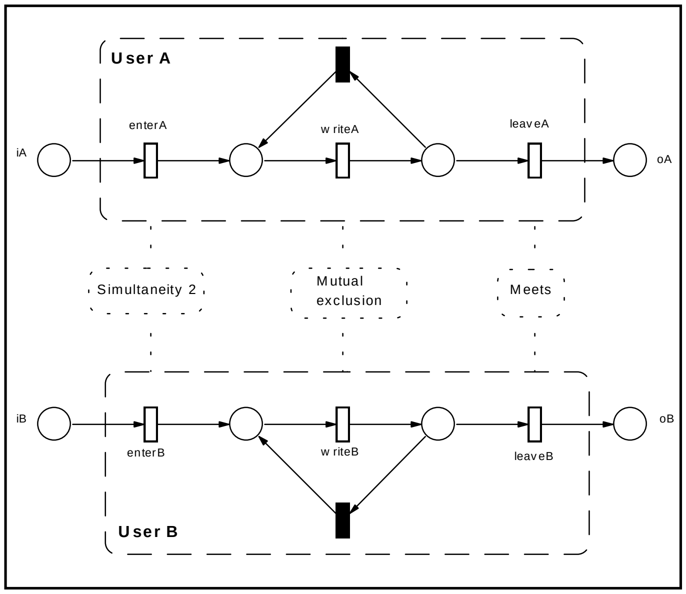

Figure4: Workflow level of the example.

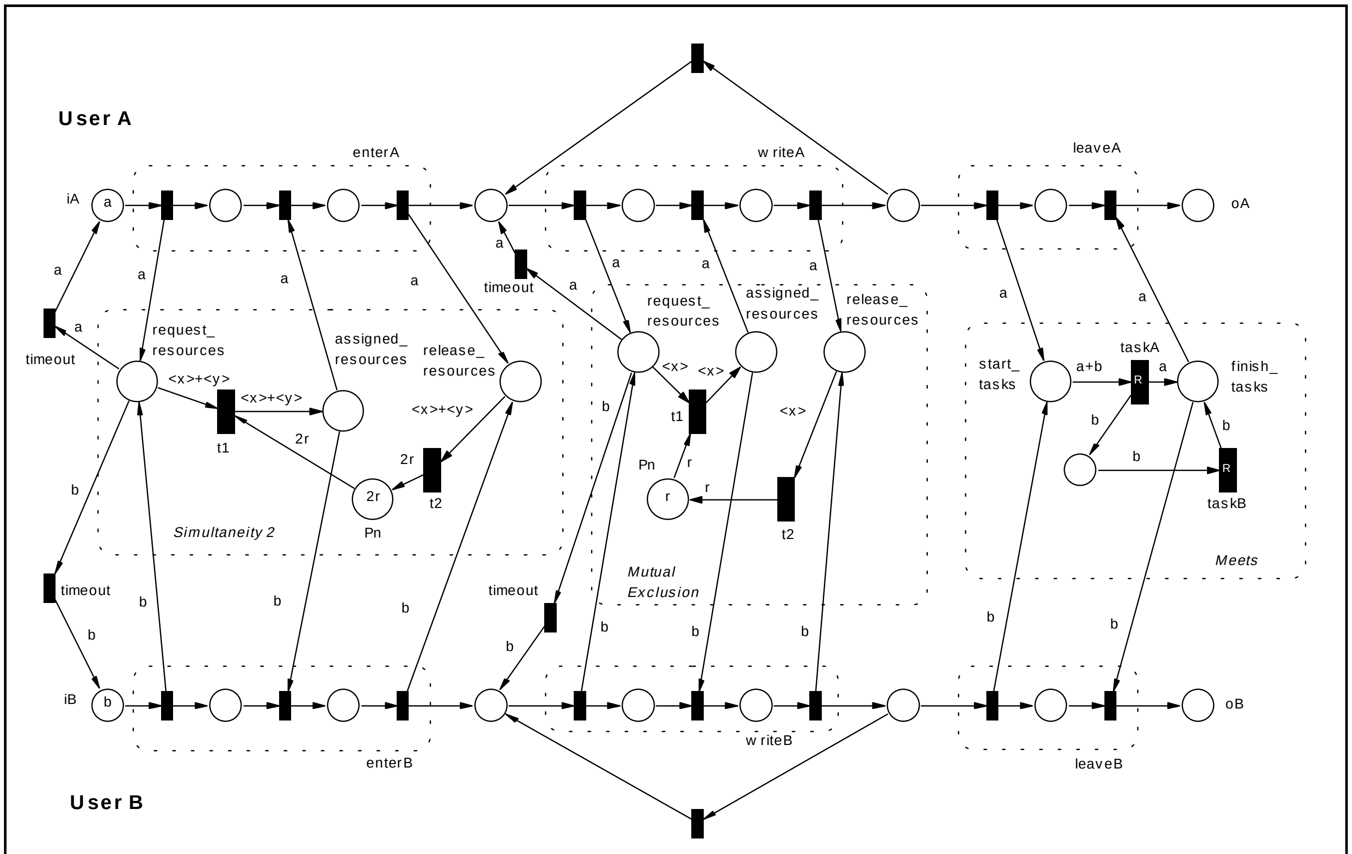

Figure 5: Coordination level of the example.

It is necessary to observe that interdependent tasks of Figure 5 are not expanded exactly according to the model of Figure 2. The reason is that in the case of temporal dependencies, only places *start_tasks* and *finish_tasks* are used, and in the case of resource management dependencies, only *request_resources*, *assigned_resources* and *release_resources* are used. Therefore, we used simplified versions of the model.

The coordination level model of the environment can be simulated and analyzed with any tool that supports high level PNs (see [5] for a list of PN simulation tools).

Verification and validation analysis indicated that our examples has eight final states. Seven of them can be eliminated by the correct use of timeouts (i.e., with adequate waiting times). The eightieth final state is the correct one (tokens in *oA* and *oB*). Performance analysis could be realized, for instance, to measure average waiting times of an user, given the rates with which the other requests the resources.

Finally, it is necessary to reinforce that the example represents just a hypothetical situation. We did not stress all details for the modeled scenario. Its main goal was to show how the coordination mechanisms can be used in a practical situation.

6. CONCLUSION

The coordination of interdependent activities in collaborative environments is a problem that should be addressed to ensure the effectiveness of the cooperation. The separation between activities and dependencies, and the utilization of reusable coordination mechanisms are steps towards this goal.

Petri nets, due to their support for modeling, simulation and analysis, have proven to be a powerful tool for verifying the correctness and validating the effectiveness of collaborative environments before their actual implementation [2], [12]. Furthermore, the hierarchical description of PNs showed an appropriate way to define the coordination structure in different abstraction levels (workflow and coordination levels). In particular, high level PNs also reduces the problem of state explosion.

The set of interdependencies presented in this paper does not claim to be complete. It would be very difficult to establish a framework of all possible interdependencies. For that reason, we opted for an extensible approach. When a new kind of interdependency arises, a corresponding coordination mechanism can be modeled and easily inserted between corresponding tasks.

One of the next steps of this work is to automate the passage from the workflow to the coordination level of models in a high level PN simulation tool. We have already done this for ordinary PN models [12].

Due to their generality, the presented coordination mechanisms are adequate to a wide range of collaborative systems, from interorganizational workflows to virtual environments. Presently, we are implementing the mechanisms to be used in the development of collaborative virtual environments. The coordination of activities will facilitate the use of this kind of environment for the realization of tasks that cannot be controlled by the social protocol.

7. REFERENCES

[1] W. M. P. van der Aalst. Modelling and analysing workflow using a Petri-net based approach. *Proc. 2nd Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pp. 31-50. 1994.

[2] W. M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1): 21-66. 1998.

[3] J. F. Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, 23: 123-154. 1984.

[4] W. Brauer, W. Reisig and G. Rozenberg (Eds.). *Petri Nets: Central Models and Their Properties – Advances in Petri Nets 1986*. Lecture Notes in Computer Science, 254. Springer-Verlag, 1986.

[5] CPN Group, Univ. of Aarhus, Denmark. *The World of Petri Nets – Tools on the Web*. 2000. <http://www.daimi.aau.dk/~petrinet/tools>

[6] F. De Cindio, G. De Michelis and C. Simone. *The Communication Disciplines of CHAOS*. In *Concurrency and Nets*, pp. 115-139. Springer-Verlag, 1988.

[7] R. Furuta and P. D. Stotts. Interpreted Collaboration Protocols and their use in Groupware Prototyping. *Proc. of the Conf. on Computer Supported Cooperative Work (CSCW'94)*, pp. 121-131. 1994.

[8] D. Gelernter and N. Carriero. Coordination Languages and their Significance. *Communications of the ACM*, 35(2): 97-107. February 1992.

[9] A. W. Holt. *Coordination Technology and Petri Nets*. In G. Rozenberg (Ed.). *Advances in Petri Nets*, pp. 278-296. Lecture Notes in Computer Science, 222. Springer-Verlag, 1985.

[10] D. Li and R. Muntz. COCA: Collaborative Objects Coordination Architecture. *Proc. of the Conf. on Computer Supported Cooperative Work (CSCW'98)*, pp. 179-188. 1998.

[11] T. Murata. Petri Nets: properties, analysis and applications. *Proc. of the IEEE*, 77(4): 541-580. 1989.

[12] A. B. Raposo, L. P. Magalhães and I. L. M. Ricarte. Petri Nets Based Coordination Mechanisms for Multi-Workflow Environments. To be published in the *Int. J. of Computer Systems Science & Engineering*, September 2000.

[13] T. Winograd and F. Flores. *Understanding Computers and Cognition: A New Foundation for Design*. Ablex, 1986.

Acknowledgements: The first author is sponsored by FAPESP (Foundation for Research Support of the State of São Paulo), process number 96/06288-9. We also would like to thank DCA – FEEC – Unicamp for the support granted to this research.