International Journal of Computer Systems Science & Engineering, v. 15, n. 5, p. 315-326, September 2000. CRL Publishing, U.K.

Petri Nets Based Coordination Mechanisms for Multi-Workflow Environments

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 cooperative workflows occurs in parallel to the definition of a common communications infrastructure among organizations. In this paper, we present a library of coordination mechanisms modeled with Petri Nets. These mechanisms specify and control the interaction between workflow processes. The separation between activities and dependencies, managed by the coordination mechanisms, allows the reuse of these mechanisms in other environments and also the use of different coordination policies in the same environment.

1. Introduction

The increasing globalization of the world economy offers promising opportunities for the field of interorganizational workflows, which cross a single organization boundary, being composed of several organizations working cooperatively. However, the necessity of cooperation raises many problems which we can separate into two main classes, those related to the definition of a joint communications infrastructure and those related to the coordination of cooperative workflows.

The first class of problems deals with the integration of heterogeneous software environments in a common communications infrastructure. These problems can be roughly summarized by the difficulties of establishing an "interoperability contract" [1] among the cooperative organizations. This "contract" must establish, among other things, which workflow engines within an organization are capable of interoperating with which engines within other organizations, the transport technology, the communication protocol, security policies and exception handling. The Workflow Management Coalition (WfMC) is currently working on standards addressing these issues [2].

The other class of problems – those related to the coordination of cooperative workflows – appears in a higher abstraction level and, to our knowledge, it has not been addressed by the WfMC yet. Here, it is assumed that the communications environment is well-defined by the partners and the major concerns are related to the coordination of interdependent activities. Since this kind of problem is the main focus of the paper, we purposely use the term "multi-workflow environment" instead of the standard "interorganizational workflow".

Coordination can be defined as "the act of managing interdependencies between activities performed to achieve a goal" [3]. The coordination is highly dynamic, because it needs to be "renegotiated" almost continuously during a collaborative effort. The great challenge in proposing coordination mechanisms to control a cooperative activity is to achieve the flexibility demanded by the dynamism of the interaction

among the partners [4]. In other words, coordination policies vary according to the cooperation instance, or even during the evolution of the same instance. Therefore, it is essential that coordination mechanisms be flexible enough to handle these variations. It is not reasonable to think that organizations will spend time and money to establish a common communications infrastructure with their partners if they are not sure about the effectiveness of the coordination structure of their cooperative workflows.

This paper presents a library of coordination mechanisms that can be reused in several workflow environments. The main idea is to separate the coordination mechanisms from the tasks that constitute the workflow. The separation between activities (tasks) and dependencies (managed by the coordination mechanisms) allows the use of different coordination policies in the same environment by changing only the coordination mechanisms. Moreover, the same mechanisms can be used in different single- or multi-workflow environments.

To model the proposed coordination mechanisms, we use an approach based on Petri Nets (PNs). The graphical representation of PNs, besides being easy to understand, enables detail encapsulation, offering a very clear hierarchical description model, which is adequate to model the different coordination levels. Furthermore, PNs offer a strong theoretical support for the analysis of an environment's behavior and supplementary simulation techniques. With the PN based model, it is possible to anticipate and test the behavior of multi-workflow environments before their implementation.

This paper is structured as follows. The next section briefly introduces PNs. Section 3 makes an overview of previous works regarding the use of PNs in coordination, workflow applications and interorganizational workflows. The library of coordination mechanisms is presented in Section 4, and an example of use is shown in Section 5. The last section presents the conclusions and next issues.

2. Petri Nets Fundamentals

PNs [5], [6] are a modeling tool applicable to a variety of fields and systems, specially suitable for systems with concurrent events. Formally, a PN can be 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 marks (also called 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 very useful notation for PNs is the graphical notation (Figure 1) which will be used in the examples throughout this paper. In this notation, 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 the PN of Figure 1, only transition *t2* is enabled; *t1* is not enabled because it would require two tokens in *P1* to fire, since *w(P1,t1)=2*. When *t2* is fired, the tokens in *P2* and *P3* are removed and *P4* receives one token. Note that the number of tokens in a PN is not necessarily conserved.

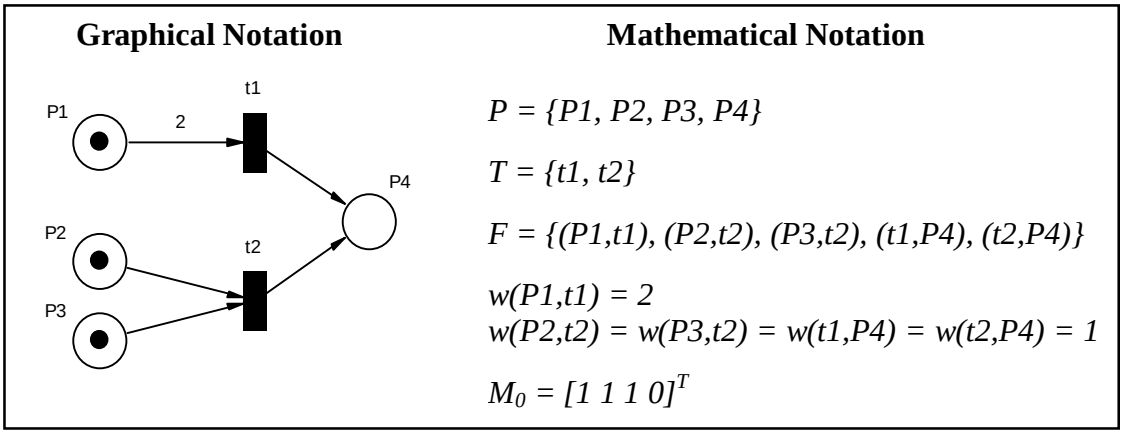

Figure 1: PNs graphical and mathematical notations.

In addition to the basic PN model, several extensions appears in the literature [6]. In this paper we use two extensions: inhibitor arcs and nets with time. 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. The basic PN model does not consider the notion of time. One way to include this notion in the model is to establish a waiting time for the tokens in a place before they enable the output transitions [7]. Time can also be associated with transitions firing. In this case, tokens do not stay in input places waiting for the firing, but they are removed from that places and some time later (firing time) are added to the output places. This kind of non-instantaneous firing is called firing with token reservation.

Besides the modeling capabilities of PNs, their support for analysis is very important and useful on the verification of workflows. "The abundance of available analysis techniques shows that Petri nets can be seen as a solver independent medium between the design of the workflow process definition and the analysis of the workflow" [8]. These analysis techniques are based on the properties of the mathematical model. Some of these properties are:

Reachability: is there a sequence of firing that reaches a given state? The coverability tree, that offers a vision of the complete sequence of transitions and states in a PN, can be used to verify this property.

Liveness: is there any state or sequence of states which will not be reached anymore, indicating a possible deadlock?

Reversibility: is it possible to return to a defined initial state M_0?

Boundness: will a place be overloaded? A PN is defined as *k*-bounded if the number of tokens in each place does not exceed *k*.

Persistence: is the firing of any pair of enabled transitions interdependent, i.e., the firing of one will disable the other? A pair of transitions with interdependent firings indicates a conflict, because only one of them will be fired (logical OR).

Three kinds of analysis can be applied to a PN model; verification, validation and performance analysis [8]. 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.

Summarizing, PNs provide interesting modeling characteristics, a strong theoretical support for analysis and also a number of simulation techniques, which make them a powerful tool for workflow applications. They enable designers to preview and test the workflow behavior before implementing it.

3. Related Work

In 1985 Anatol Holt used PNs to coordinate activities in computer supported work environments [9]. He proposed a new interpretation for PNs, connecting their formal structure with the "natural structure" of human (or computational) work. Moreover, he also identified some essential aspects of coordination technology, which at that time was very promising for the creation of electronic work environments. Among these aspects, the most important was related to the flexibility of coordination mechanisms. As stated by Holt, to be useful, the desired patterns of task coordination "must be done in a flexible yet well-integrated manner, with plenty of leeway for the unpredictability of real life" [9].

Holt's work resulted in the creation of Diplan, a formal graphical language to the planning of activities involving multiple collaborative agents [10]. This language is based on Predicate/Transition Nets [11], with some alterations, such as the specification of a time for state changes (non-instantaneous transitions) and the explicit reference to the human role in the tasks' execution. The CHAOS (Commitment Handling Active Office System) is another system for the coordination of activities in office automation based on PNs [12].

PNs also constitute the basis of Trellis, a model for prototyping interaction protocols in collaborative systems [13]. The generated protocol defines how a central server must process clients' requests. Trellis uses colored PNs [14], in which tokens have a type (color) and carry information. The notion of time appears in the form of delays and timeouts to the firing of transitions. The functionality of Trellis is different from that of Diplan because the former is not restricted to the coordination of activities, creating "hyperprograms" that join hypermedia navigation to collaboration support.

The ideas related to the automation and coordination of tasks led up to the development of workflow systems [15], [16], defined as "a particular kind of groupware intended to assist groups of people in executing work procedures; they contain knowledge of how work normally flows through the organization" [17]. PNs and their extensions are also used for the modeling of this kind of system. There are many papers in the literature presenting the mapping of workflow concepts into PNs (e.g., [17], [18]). Basically, tasks are represented by transitions and jobs are represented by tokens flowing through the net. The state of a job is given by the marking of the PN at a given instant. Control activities are also represented by transitions, which are used in basic workflow connections such as joins and splits. Examples of variations of PNs used in workflow applications are the ICNs (Information Control Nets) [17] and the Workflow Nets (WF-Nets), a class of PNs appropriate to the representation, validation and verification of sets of interdependent tasks [8].

PNs are not the only way to evaluate workflow systems before implementing them. Current workflow tools offer approaches less formal than PNs for analysis, which have a larger penetration in the business world [19]. SIMPROCESS [20] is an example of a commercial tool that uses workflow based simulation and analysis to evaluate alternatives prior to implementing them.

Recently, interorganizational workflows have attracted a lot of attention from developers. PRODNET II [19] is an example of a project aiming to develop an open

and flexible support infrastructure suited to the needs of small and medium enterprises. The PRODNET II architecture implements a clear separation between support services and control mechanisms, and adopts a workflow based coordination approach.

METEOR (Managing End-To-End OpeRations) [21] is a well-succeeded academic workflow management system that generated a commercial product. METEOR specifies human and automated tasks and intertask dependencies, generating automatic code from graphical specification. Communication channels between heterogeneous and distributed workflow engines are provided by means of a CORBA and Java Web-based implementation.

PNs have also been used in interorganizational workflows. An example is the virtual machine (a PN) that controls the behavior of mobile agents in COSM (Common Open Service Market) [22]. COSM uses mobile agents to support the management of interorganizational workflows, an approach specially suitable for situations where partners do not intend to tighten their cooperation (e.g., in order to make a price survey with several suppliers and get the best offer). This is not the case covered by this paper; here we assume that partners have a well-established set of tasks in the cooperation.

To our knowledge, the work more closely related to ours is that of van der Aalst [23], who realized that it is necessary to address the contents of the coordination structure of interorganizational workflows. As stated by him "the semantics of the constructs needed to model interorganizational workflows should be defined before solving the technical issues (mainly syntactical)" [23]. He expanded his previous work on WF-Nets [8] in order to model and analyse interorganizational workflows.

Although being motivated by similar concerns and making use of the same modeling tool (PNs), there are differences between his work and ours. The main goal of van der Aalst's work is to formally verify the correctness of interorganizational workflows and their consistency with the message sequence charts used to specify the interaction between workflow processes. Our main goal, on the other hand, is to provide a set of PN-modeled coordination mechanisms that can be used to specify such interaction in several workflow environments. The correctness and consistency of the generated workflow can be verified by means of available PN analysis and simulation tools. In order to facilitate this task, we also have developed a prototype tool capable of "inserting" those mechanisms between interdependent tasks of a PN modeled with a specific software tool.

4. A Library of Coordination Mechanisms

As already stated, this work intends to provide mechanisms to manage interdependencies between tasks in a multi-workflow environment. In order to achieve this goal, we propose a library of PN-based primitives to model coordination mechanisms for several possible interdependencies. Each coordination mechanism ensures that the associated dependency will not be violated. The idea is that workflow designers be concerned only with the definition of tasks and interdependencies among them, and not with the management of those dependencies, since mechanisms to manage them are provided by the library.

In the proposed schema, an environment is modeled in two distinct levels, *workflow* and *coordination*. In the *workflow* level, each workflow is modeled separately by a PN, in which tasks are represented by transitions, and that may include "traditional" workflow connections (splits and joins) and routings (parallel routing, conditional routing, etc). Also in this level, it is necessary to establish the interdependencies between tasks in different workflows, or even in the same workflow.

The *coordination* level is built under the workflow level by the expansion of interdependent tasks according to a model defined in [18] and the insertion of corresponding coordination mechanisms between them.

In the passage from the workflow to the coordination level, each task (a transition in the workflow level) which has a dependency with another is hierarchically expanded in a system with five transitions (*ta, tb, ti, tf* and *tc*) and four places (*P1, P2, P3* and *P4*), as proposed by van der Aalst et al [18]. As shown in Figure 2, attached to each expanded task there are also five places that represent the interaction with the resource manager and the agent that executes the task. The places *request_resource*, *assigned_resource* and *release_resource* connect the task with the resource manager. The places *start_task* and *finish_task* connect the task with the agent that performs it, respectively indicating the beginning and the end of the task execution.

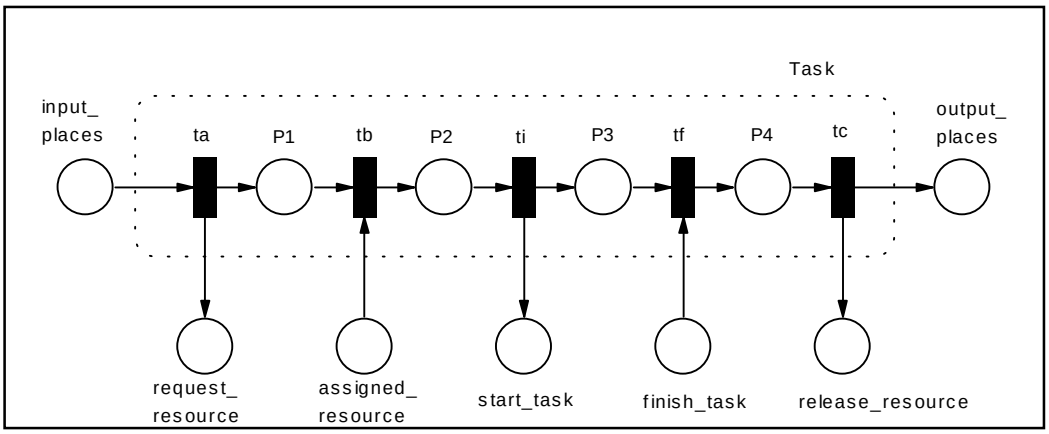

Figure 2: An expanded task in the coordination level.

Based on the model of Figure 2, it is possible to conclude that a task can be connected with two sub-nets: a resource manager (which has *request_resource* and *release_resource* as input places and *assigned_resource* as output place), and another representing the logistics of the task (which has *start_task* as input place and *finish_task* as output place).

The library recognizes two general classes of interdependencies: *temporal* and *resource management* dependencies. The former is related to the logistics sub-net cited above, while the latter is related to the resource manager sub-net.

The library construction enables to build 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.

In the following sections we present the set of mechanisms of the library.

4.1. Temporal Dependencies

Temporal dependencies establish the execution order of “disconnected tasks” (e.g., tasks belonging to different workflows). By the use of coordination mechanisms proposed for temporal dependencies, it is possible to define if a task is executed before, after, or during another task.

In order to define the possible temporal dependencies between tasks, we refer to a classic temporal logic paper by J. F. Allen [24]. In this paper, Allen states that there is a set of primitive and mutually exclusive relations that can be applied to time intervals:

[x1, x2] *equals* [y1, y2] iff x1 = y1 and x2 = y2. (X and Y are the same time interval.)

[x1, x2] *starts* [y1, y2] iff x1 = y1 and x2 < y2. (X and Y start together, but X finishes before Y.)

[x1, x2] *finishes* [y1, y2] iff x2 = y2 and x1 > y1. (X and Y finish together, but X starts after Y.)

[x1, x2] *before* [y1, y2] iff x2 < y1. (X happens before Y, and they do not overlap.)

[x1, x2] *meets* (y1, y2] iff x2 = y1. (X happens before Y, which starts immediately with the end of X.)

[x1, x2] *overlaps* [y1, y2] iff x1 < y1 < x2 < y2. (X starts before Y, which starts before the end of X.)

[x1, x2] *during* [y1, y2] iff x1 > y1 and x2 < y2. (X is totally contained in Y.)

Allen defined possible relations between time intervals. We adapted these relations for the definition of temporal dependencies between tasks in multi-workflow environments, adding a couple of new relations and a few variations of those originally proposed. The following temporal dependencies are defined in the library:

task1 equals task2: this dependency establishes that both tasks must be executed simultaneously. In the coordination level, it is necessary to ensure that tasks only start when both are ready (tokens in places *start_task* – Figure 2) and that they finish together (tokens simultaneously sent to place *finish_task*).

task1 starts task2: according to Allen's definition, both tasks must start together and task1 must finish before task2 (we called this relation *startsA*). However, 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 ([x1, x2] *startsB* [y1, y2] iff x1 = y1).

task1 finishes task2: the original definition establishes that both tasks must finish together and task1 must start after task2 (*finishesA*). Similarly to the previous dependency, it is possible to relax the definition, creating a variation in which it does not matter which task starts before ([x1, x2] *finishesB* [y1, y2] iff x2 = y2).

task2 after task1: this relation derived from the original relation *before*. In this case, there is a restriction on the execution of task2, which can only occur after the execution of task1. There is no restriction on the performance of task1. This relation models the notion of pre-requisite, frequently used in workflows. It is possible to define two variations of this relation. The first one (*afterA*) establishes that each execution of task1 enables a single execution of task2. The second variation (*afterB*) enables many executions of task2 after a single execution of task1.

task1 before task2: differently from the previous relation, the restriction here occurs on the execution of task1, which cannot be performed if task2 has started to execute. There is no restriction on the execution of task2 (i.e., task2 does not have to wait for the execution of task1, as was the case for *task2 after task1*). From the temporal logic point of view, the difference between these two relations may not be significant, but it generates totally different coordination mechanisms.

task1 meets task2: according to this dependency, task2 must begin immediately with the end of task1. In the coordination level, we achieve this by blocking the end of task1 (token in *finish_task* – Figure 2) while task2 is not ready (token in *start_task*).

task1 overlaps task2: this dependency generates two different models. The first one (*overlapsA*) follows the original definition, which establishes that task2 must start before the end of task1, which must finish before task2. The second model relaxes the last part of the original definition and does not require that task1 finishes before task2 ([x1, x2] *overlapsB* [y1, y2] iff x1 < y1 < x2).

task2 during task1: this dependency establishes that task2 must be executed during the execution of task1. Once more, the relation enables two different interpretations, which generates two different coordination mechanisms. In the first case (*duringA*), task2 can be executed only once during the performance of task1. In the second case (*duringB*), task2 can be executed several times.

In the following, we show the models for the coordination mechanisms related to *starts* and *during* dependencies. Other mechanisms are shown in the Appendix and the full set of coordination mechanisms is described in [25].

Figure 3 shows the coordination mechanism for the relation *Task1 startsB Task2*, without the restriction on which task should finish before. In the figure, transition *t1* is a control activity that constitutes, at the same time, an AND-join and an AND-split, ensuring the simultaneous beginning of both tasks. The transitions called *task1* and *task2* represent the logistic of the tasks. They are represented as non-instantaneous transitions with token reservation (indicated by the letter "R" in the transitions). In order to model the relation with the restriction that Task1 must finish before Task2 (*startsA*), it is necessary to add an AND-join after the tasks, ensuring that a token will be sent to place *finish_task2* only after the end of both tasks (Figure 4).

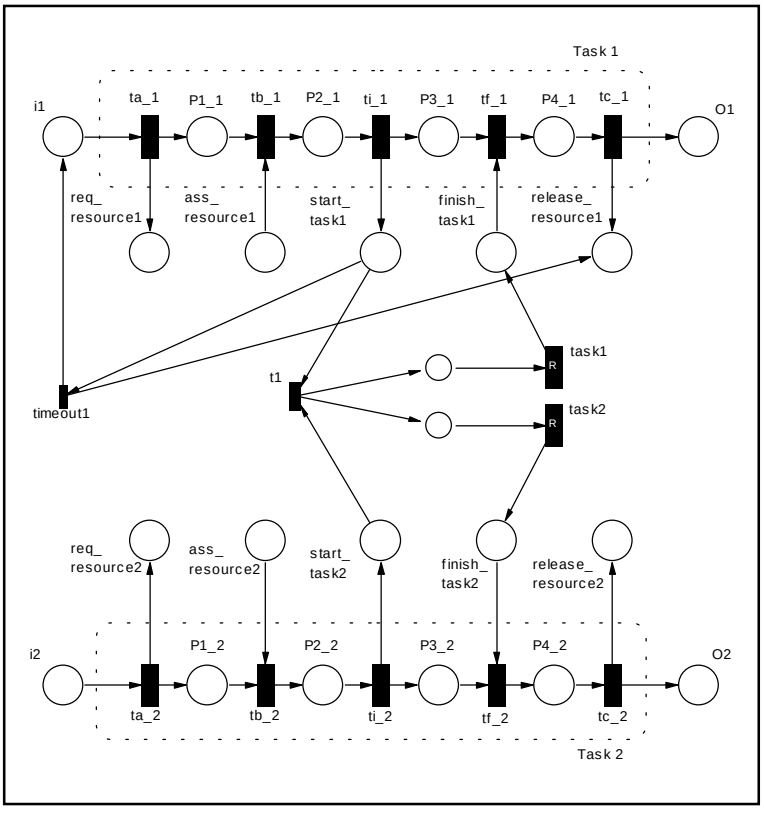

Figure 3: Coordination mechanism for *Task1 startsB Task2*.

Since the goal of the coordination mechanisms is to deal with relations between tasks that sometimes belongs to complex procedures, it is interesting to add mechanisms to reduce deadlocks. For example, in relation *Task1 startsB Task2*, the first workflow

could be blocked in Task1 if the second workflow has an alternative path that does not execute Task2. This kind of problem can be minimized by the use of timeouts added to the original schema.

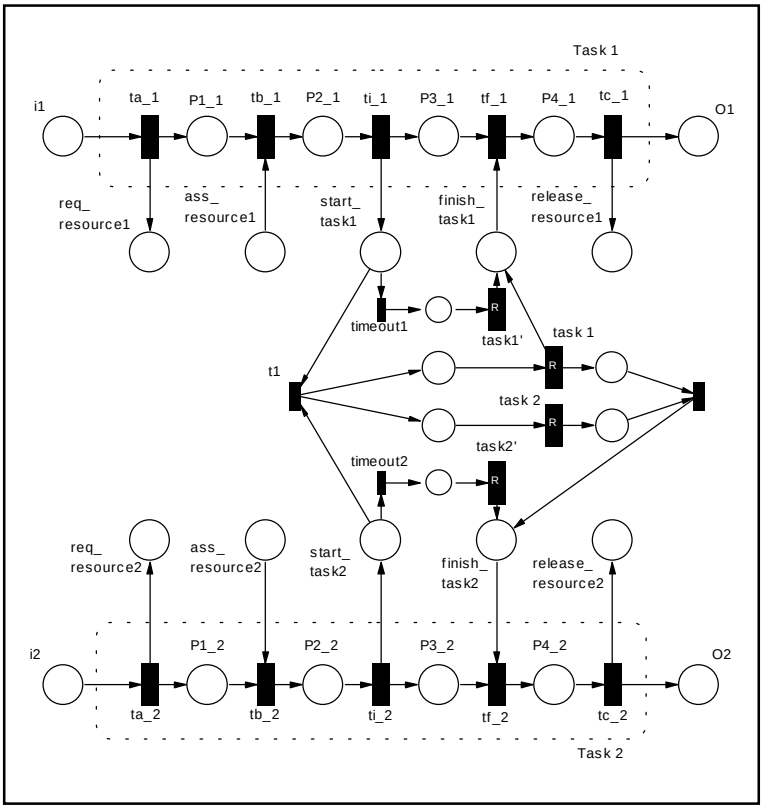

Figure 4: Coordination mechanism for *Task1 startsA Task2*.

Two kinds of timeouts are proposed. In the first kind (called *timeoutA*), an alternative task (*task'*) is executed after a certain waiting time if the original task has not been executed yet. The other kind of timeout (*timeoutB*) is modeled by a timed transition that removes the token from *start_task* and returns it to the input places of the task after a certain waiting time. This enables to follow alternative paths that do not execute the blocked task. In this kind of timeout, it is also necessary to put a token in *release_resource* if the task have requested a specific resource. An example of *timeoutB* is shown in Figure 3 for Task1 (transition *timeout1*) and examples of *timeoutA* are shown in Figure 4 for both tasks.

The coordination mechanism for the relation *Task2 duringA Task1* is shown in Figure 5. In the model, the firing of *t1* sends tokens to places *P1* and *P2*, the latter indicating to Task2 that Task1 has already started (enabling the firing of *task2* only once). After the firing of *task1* (token in *P3*), a token will only be sent to *finish_task1* at the end of Task2 (token in *P4*). In order to avoid that Task1 waits indefinitely, there is a timeout inhibited by the presence of a token in *start_task2* (i.e., Task1 will not be finished if Task2 is ready to begin).

In order to enable multiple executions of Task2 during the execution of Task1 (*duringB*), it is necessary to make a few modifications in the model of Figure 5. The first one is to add a return arc from *task2* to *P2*, enabling future firings of *task2*. In addition, transition *t2* must have *P2* as input place instead of *P4*, which is removed from the model. The reason for this is that, at the end of Task1 (firing of *t2*) the token must be removed from *P2* to avoid future occurrences of Task2. It is also necessary to

include an inhibitor arc from *start_task2* to *t2*, avoiding that Task1 finishes while Task2 is ready to be executed. The resulting model is presented in Figure 6.

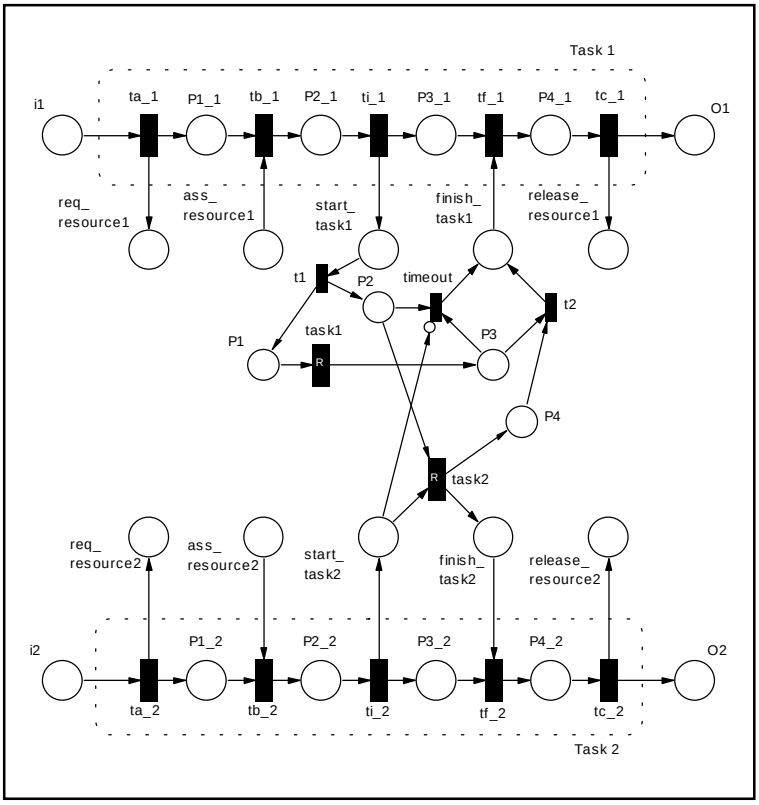

Figure 5: Coordination mechanism for *Task2 duringA Task1*.

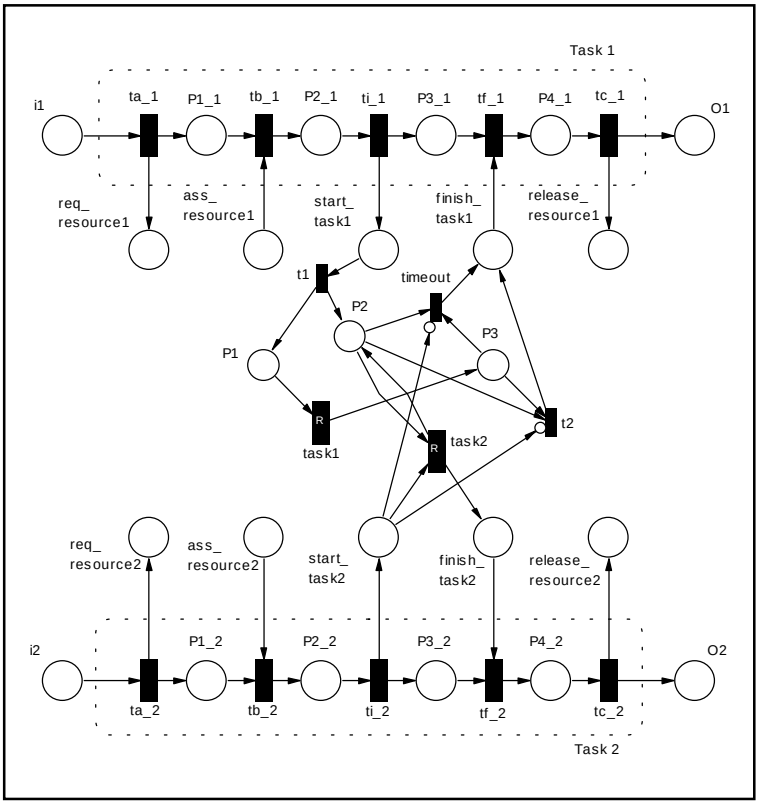

Figure 6: Coordination mechanism for *Task2 duringB Task1*.

In both cases (*duringA* and *duringB*) there is an internal timeout to avoid that Task1 waits indefinitely for Task2. In other words, Task1 may be skipped if it takes too long to starts its execution.

The coordination mechanisms presented in this section cover only those dependencies related to the synchronization of tasks. Dependencies related to the use of resources are modeled by another class of mechanisms presented in the following section.

4.2. Resource Management Dependencies

The 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. It is necessary to clarify that the term "resource" in this context refers not only to the agent that performs the task (similar to the concept of actor [17]), but also to any artifact needed to the execution of the task (e.g., a pen in a shared whiteboard interaction).

We define three basic coordination mechanisms for resource management:

Sharing: a limited number of resources needs to be shared among several tasks. It represents a very common situation that occurs, for example, when various computers share a printer.

Simultaneity: a resource is available only if a certain number of tasks requests it simultaneously. It represents, for instance, a machine that can only be used with more than one operator.

Volatility: indicates whether, after the use, the resource is available again. For example, a printer is a non-volatile resource, while a sheet of paper is volatile.

Differently from temporal dependencies, resource management dependencies are not binary relations. It is possible, for example, 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).

Figure 7 shows the coordination mechanism for the simultaneity, which assigns a resource only if *N* tasks request it simultaneously. The model has a place *Pn* with *N* tokens (in the figure, *N=2*, as indicated by the two tokens in *Pn*) that return to this place at the end of the tasks (via *release_resource*). Place *Pn* is connected to transition *t1* by an arc with weight *N*. Transition *t1* fires only when there are *N* tokens in *Pn* (indicating *N* available instances of the resource) and also *N* tokens in *P1* (indicating that N tasks have already requested the resource). After the firing, *t1* sends *N* tokens to *P2*, which distributes them among the *N assigned_resource* places. Places *P3* and *P4* are used to indicate which tasks have requested the resource and to avoid that the same task requests the resource more than once (i.e., the resource must be assigned to *N* distinct tasks).

The library also defines composite mechanisms from the basic ones discussed above.

S*haring M + simultaneity N*: represents the situation in which up to *M* groups of *N* tasks can share a resource.

Sharing M + volatility N: situation in which up to *M* tasks can share the resource, which can be used *N* times.

Simultaneity M + volatility N: the resource is assigned to groups of *M* tasks simultaneously. This can be done *N* times.

Sharing M + simultaneity N + volatility Q: up to *M* groups of *N* tasks can share a resource. This can be done *Q* times.

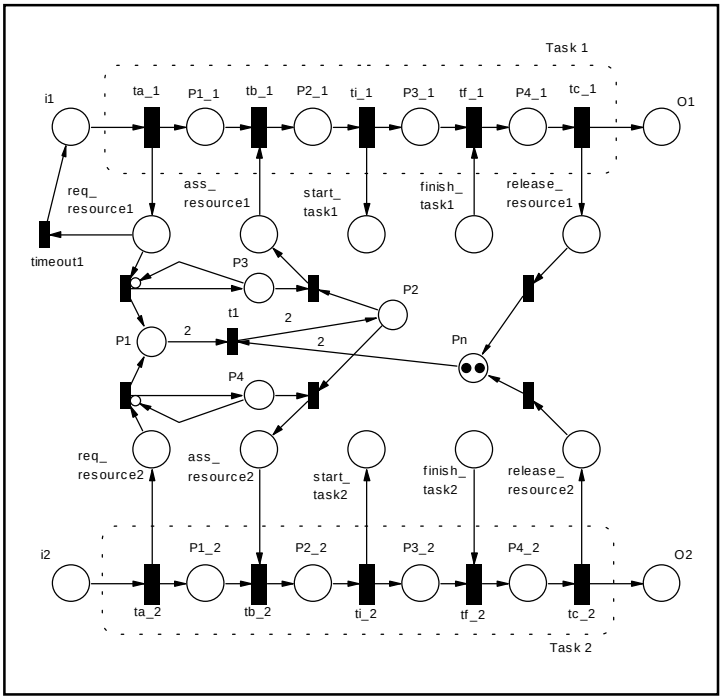

Figure 7: Resource manager for *simultaneity* (*N=2*), with two tasks able to request the resource.

Similarly to temporal dependencies, it is possible to define timeouts for the tasks in order to reduce deadlocks. In this case, there is a single type of timeout, which is modeled by a timed transition from *request_resource* back to the input places of the task (see Figure 7).

4.3. A Prototype Implementation

In order to automate the passage from the workflow level to the coordination level of the models, by the expansion of tasks and the insertion of coordination mechanisms, we implemented an application based on three components: a PN simulation tool, a language for the definition of interdependencies among tasks, and a program capable of generating the PN in the coordination level (Figure 8).

The specific language for the description of PNs (input files) normally varies according to the simulation tool used. However, the language for the definition of interdependencies is independent of the simulation tool. The file with the interdependencies define, one at each line, all the dependencies among tasks in the workflow level. It is necessary that tasks in this file have the same name as the transitions representing them in the workflow level. Otherwise, the program will not be able to know where to insert the coordination mechanisms. Basically, a dependency is defined as `<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. As an example, the simultaneity dependency shown in Figure 7 is described as `sim 2 “task1” “task2” time_out`. In this example, parameter N=2 and only the first task has a timeout.

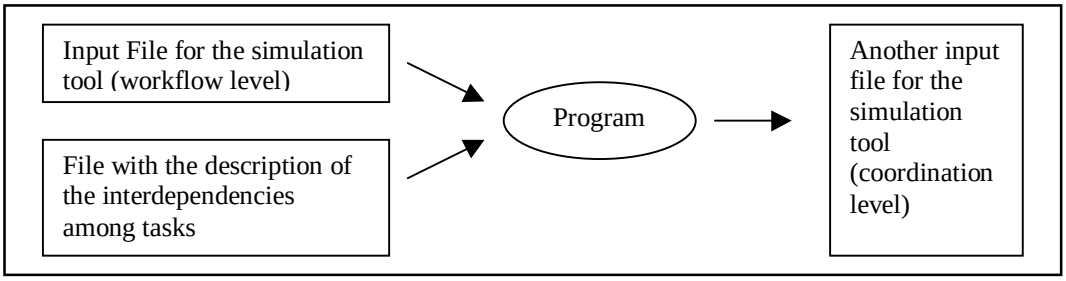

Figure 8: Schema for the use of the library with a PN simulation tool.

The implemented prototype uses the Visual Simnet [26] as simulation tool. This is a freeware tool, capable of modeling and analysing conventional and stochastic PNs (whose firing times are given by probability functions). It has a graphical editor and several resources for the analysis of PNs, including animated simulation, coverability tree, structural and performance analysis, among others. Besides its good analysis capacity, Visual Simnet was chosen because it has a very simple textual format for the description of PNs (MoDeL – Model Description Language) and it allows the exportation for other tools (e.g., INA – Integrated Net Analyzer [27], a tool with further analysis capabilities[1]).

The result of the application is a file in MoDeL describing the coordination level of the defined model, which can be simulated, analysed and validated using the Visual Simnet or a tool to which it can be exported. In the next section we present some analysis results derived from the use of this tool.

5. Example

To illustrate the use of the library, we present an example of a hypothetical multi-workflow environment composed of three independent "organizations", a consumer, a shop and a producer. This environment tries to represent a typical situation in e-commerce, in which a consumer contacts a virtual shop to buy some goods. The shop, however, is just an intermediary between consumer and producer, having to contact the latter to receive the goods that will be delivered to the consumer. The workflow level of this environment is shown in Figure 9, stressing aspects of the relation between the shop and the producer.

The consumer's workflow is very simple. It must start the process by contacting the shop to make the order, and then wait for the goods or cancel the order. However, the order can only be canceled if the product has not been delivered by the shop. This defines the relation *cancel_order (consumer) before deliver_goodsS (shop)*. The shop, after receiving an order, starts two parallel activities: verification of the consumer's credit card and contact with the producer. This contact *starts* the process in the producer. If there is any problem with the consumer's credit card, the shop cancels the order. Otherwise, the shop is able to schedule the delivery of the product, a task that must occur simultaneously to the scheduling made by the producer (*schedule_deliveryS equals schedule_deliveryP*). Note that, if there is a problem with the credit card, *schedule_deliveryS* will not be executed, and consequently neither will *schedule_deliveryP*. Using a timeout in task *schedule_deliveryP*, the producer has the option to follow an alternative path that cancels the production. Finally, there is also the relation that the shop receives the goods *after* the producer delivers them.

[1] INA does not handle transitions with token reservation. Therefore, some coordination mechanisms cannot be treated by this tool.

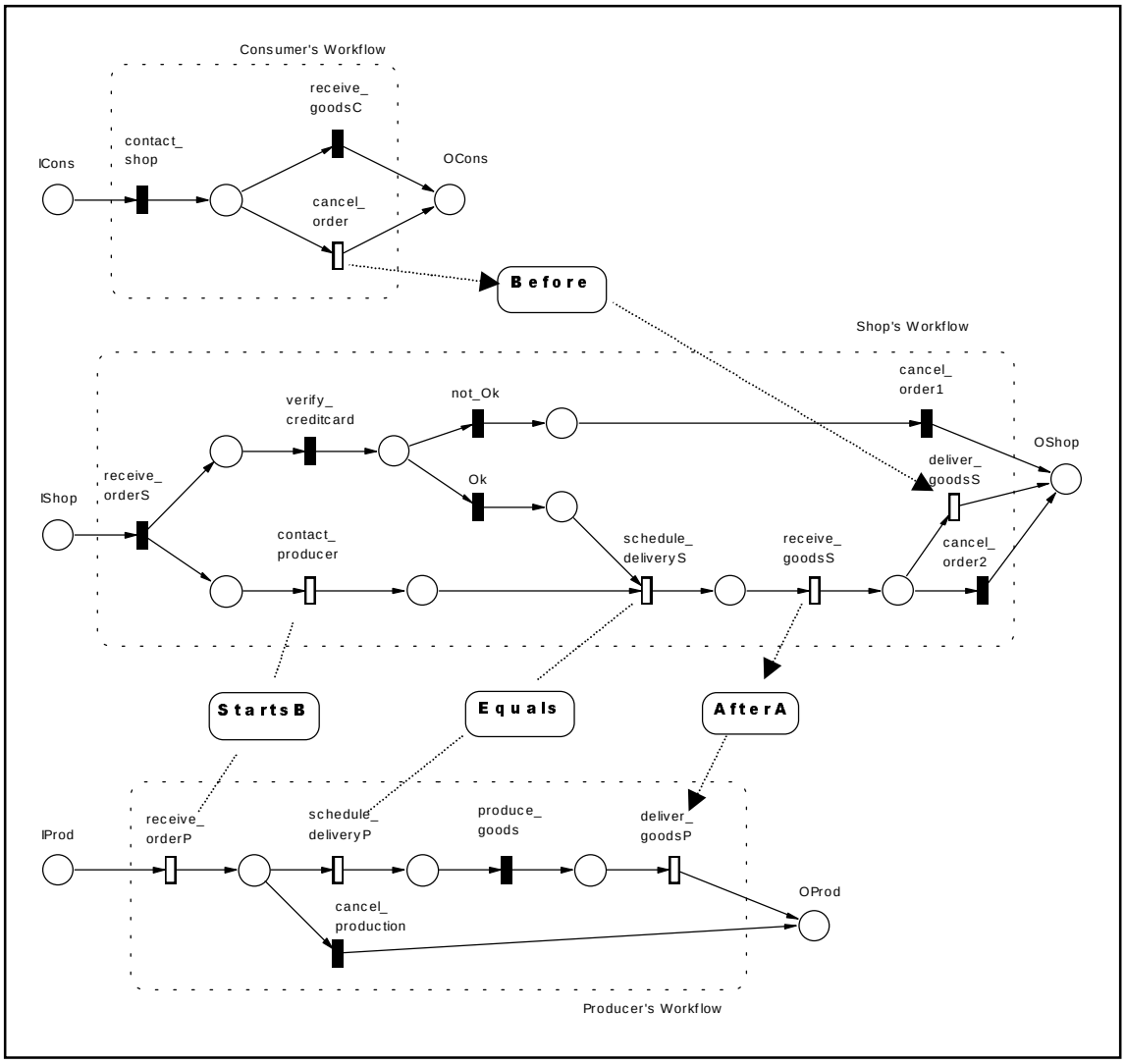

Figure 9: Workflow level of the example.

To construct the coordination level for this example using the application presented in the previous section, the following file is written to define the interdependencies among tasks (note that *cancel_order* and *schedule_deliveryP* have timeouts of type B):

```
before “cancel_order” “deliver_goodsS” time_outB
startsB “contact_producer” “receive_orderP”
equals “schedule_deliveryS” “schedule_deliveryP” no_time_out
                                                  time_outB
afterA “receive_goodsS” “deliver_goodsP”
```

The model of the coordination level for this example is shown in Figure 10. At first glance, this model may seem complicated, but it is highly modular and easily built from the model of Figure 9, by expanding interdependent tasks (open rectangles) and inserting the pre-defined coordination mechanisms.

An observation should be made regarding the expansion of tasks. As can be seen in Figure 10, interdependent tasks are not expanded exactly according to the model of Figure 2. The reason is that in the case of temporal dependencies, only places *start_task* and *finish_task* are needed, therefore, we use a simplified version of the model. A similar simplification is made for the case of resource management dependencies. The complete model of a task, as in Figure 2, is only used if it has both a temporal and a resource management dependency.

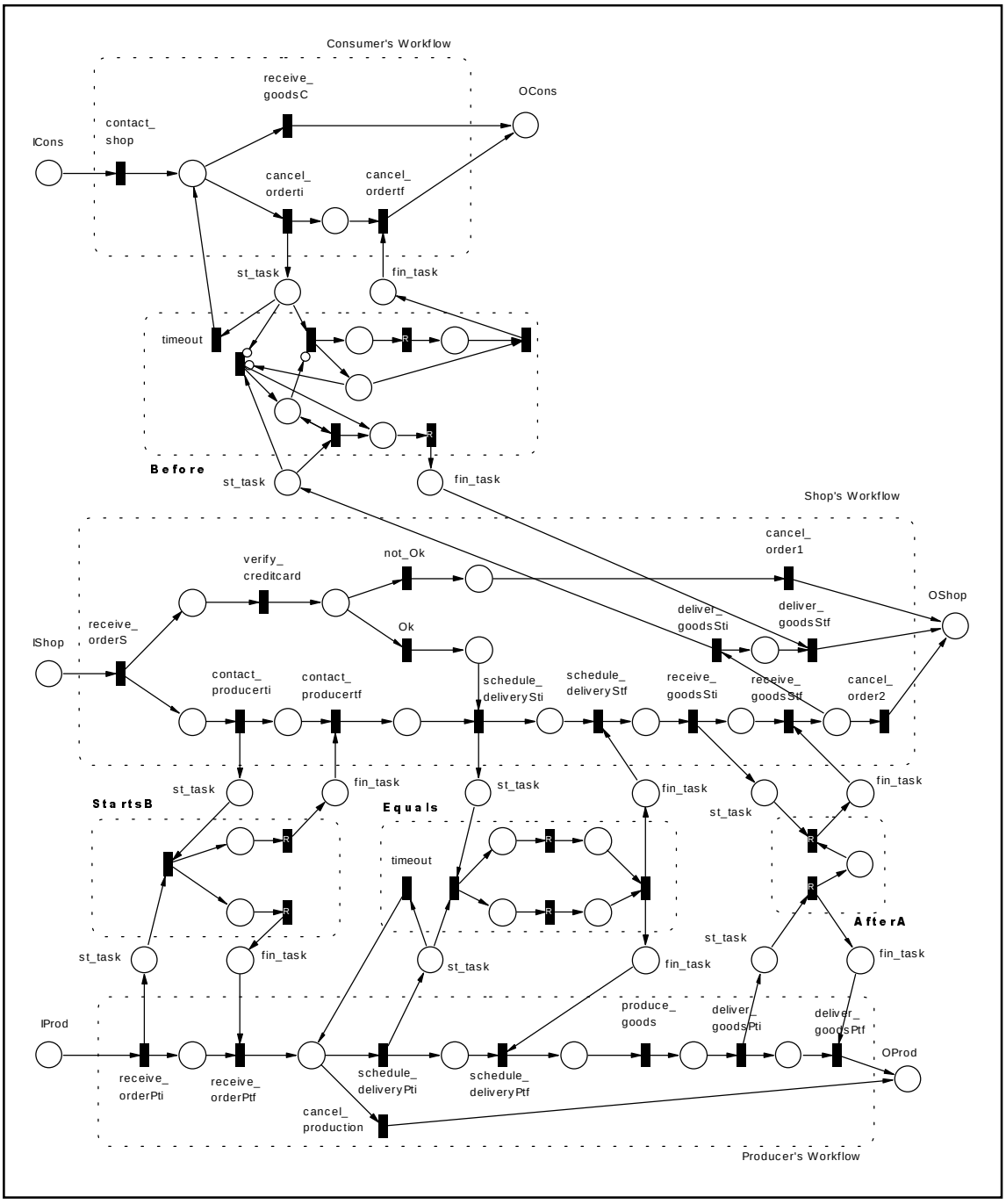

Figure 10: Coordination level of the example.

The PN of the coordination level was simulated and analysed with the Visual Simnet, validating the model. The analysis of the coverability tree presented twelve final states. Some of these final states, although correct, indicates that the model could be improved. For example, if there is a problem with the credit card, the workflows will finish correctly from the functional point of view, but there will be a "dead token" in the place between *contact_producer* and *schedule_deliveryS* in the shop's workflow, which indicates a badly-structured workflow [8]. Performance could be analysed, for instance, to measure average consumers' waiting time, given the rates with which goods can be produced.

The previous example used only temporal dependencies, but resource management dependencies could also be used in a straightforward way. For example, the shop could

have a stock, which defines an alternative route to that of contacting the producer. The task of getting the goods from the stock would have a volatility dependency, indicating that the stock would eventually finish.

It is necessary to reinforce that the presented example represents just a hypothetical situation. We did not stress all details for the modeled scenario (e.g., the consequences of the consumer's cancellation, such as refund, devolution of the goods to the producer, and so on). Its main goal was to show how the coordination mechanisms can be used in a practical situation.

6. Conclusions

This paper intends to contribute in two different ways. The first one is the definition of a generic set of interdependencies among tasks, and the second are the PN based coordination mechanisms. Due to this clear separation, non-PN based applications can also follow the ideas presented here using the interdependencies among tasks and constructing appropriate coordination mechanisms.

Petri Nets, due to their support for modeling, simulation, and analysis, has proven to be a powerful tool for verifying the correctness and validating the effectiveness of multi-workflow environments before their actual implementation [8], [17]. Furthermore, the hierarchical description of PNs showed an appropriate way to define the coordination structure in different abstraction levels (workflow and coordination levels).

Nevertheless, a typical problem in the use of PNs is the state explosion, which can occur in our context when the number of workflows and interdependencies increases. This problem can be minimized by the use of high level PNs, such as predicate/transition and colored nets. We are currently investigating how the use of these kinds of PNs can simplify the coordination mechanisms of the library [28]. We are also studying the possibility of using methodologies such as CPM (Critical Path Method) and PERT (Program Evaluation and Review Technique) [29] to concentrate the coordination effort in the critical path of complex nets.

Another use of the coordination mechanisms are in collaborative virtual environments, where remote users can be simultaneously present and interact with each other and with the objects of the simulated 3D world. In this scenarios, users can be compared to an organization in a multi-workflow environment, because their behavior is defined by an independent workflow. The collaborative interaction in the virtual world is defined by interdependencies among tasks executed by the users and coordinated by the mechanisms here presented. Therefore, a collaborative virtual environment can be viewed as a multi-workflow environment. A similar experience has been made using PNs to control computer animation [30].

The library presented in this paper does not claim to be complete. We believe it would be very difficult to establish a framework of all possible interdependencies between tasks. For that reason, we preferred to opt for extensibility instead of completeness. When a new kind of interdependency arises, a corresponding coordination mechanism can be modeled and inserted to the library, by adding a new "command" to the definition language and extending the program (Figure 8) to recognize the new dependency.

Finally, we reinforce our belief that the coordination of interdependent tasks in multi-workflow environments is a problem that should be addressed to ensure the effectiveness of the cooperation among organizations. The separation between activities

and dependencies, and the utilization of reusable coordination mechanisms are steps towards this goal.

Acknowledgements. The first author is sponsored by FAPESP (Foundation for Research Support of the State of São Paulo), grant n. 96/06288-9. This work has also been supported by the SAPIENS project (FAPESP, grant n. 97/12807-1). We also would like to thank the School of Electrical and Computer Engineering (FEEC) – UNICAMP for the expressive support granted to this research.

References

1. **Anderson, M** Workflow Interoperability – Enabling E-Commerce, *WfMC White Paper* (April 1999). <http://www.aiim.org/wfmc/standards/docs/IneropChallPublic.PDF>
2. **Workflow Management Coalition** *Interface 4 – Interoperability Abstract Specification*, WFMC-TC-1012 (October 1996). <http://www.aiim.org/wfmc/standards/docs/if4-a.pdf>
3. **Malone, T W and Crowston, K** What is Coordination Theory and How Can It Help Design Cooperative Work Systems? *Proc. ACM Conf. on Computer Supported Cooperative Work* (1990) 357-370
4. **Edwards, W K** Policies and Roles in Collaborative Applications. *Proc. ACM Conf. on Computer Supported Cooperative Work* (1996) 11-20
5. **Petri, C A** *Kommunikation mit Automaten*. Schriften des IIM Nr. 3, Bonn: Institute für Instrumentelle Mathematik (1962)
6. **Murata, T** Petri Nets: properties, analysis and applications, *Proc. of the IEEE*, **77**(4) (1989) 541-580
7. **Ramamoorthy, C V and Ho, G S** Performance evaluation of asynchronous concurrent systems using Petri Nets, *IEEE Trans. Software Engineering*, **6**(5) (September 1980) 440-449
8. **van der Aalst, W M P** The Application of Petri Nets to Workflow Management, *The Journal of Circuits, Systems and Computers*, **8**(1) (1998) 21-66
9. **Holt, A W** Coordination Technology and Petri Nets. In Rozenberg, G (Ed.) *Advances in Petri Nets*, *LNCS 222*, Springer-Verlag (1985) 278-296
10. **Holt, A W** Diplans: A New Language for the Study and Implementation of Coordination, *ACM Trans. Office Information Systems*, **6**(2) (April 1988) 109-125
11. **Genrich, H J** Predicate/Transition Nets. In Brauer, W, Reisig, W and Rozenberg G (Eds.) *Petri Nets: Central Models and Their Properties – Advances in Petri Nets 1986, LNCS 254*, Springer-Verlag (1986) 207-247
12. **De Cindio, F, De Michelis, G and Simone, C** The Communication Disciplines of CHAOS. In *Concurrency and Nets*, Springer-Verlag (1988) 115-139
13. **Furuta, R and Stotts, P D** Interpreted Collaboration Protocols and their use in Groupware Prototyping. *Proc. ACM Conf. on Computer Supported Cooperative Work* (1994) 121-131
14. **Jensen, K** Coloured Petri Nets. In Brauer, W, Reisig, W and Rozenberg G (Eds.) *Petri Nets: Central Models and Their Properties – Advances in Petri Nets 1986, LNCS 254*, Springer-Verlag (1986) 248-299
15. **Jablonski, S and Bussler, C** *Workflow Management: Modeling Concepts, Architecture and Implementation*, International Thomson Publishing (1996)
16. **Lawrence, P** (Ed.), *The Workflow Handbook 1997*, Workflow Management Coalition, John Wiley & Sons, Inc. (1997)

17. **Ellis, C A and Nutt, G J** Modeling and Enactment of Workflow Systems. In Marsan, M A (Ed.) *Application and Theory of Petri Nets 1993, LNCS 691*, Springer-Verlag (1993) 1-16
18. **van der Aalst, W M P, van Hee, K M and Houben, G J** Modelling and analysing workflow using a Petri-net based approach. *Proc. 2nd Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms* (1994) 31-50
19. **Camarinha-Matos, L M and Afsarmanesh, H** Flexible Coordination in Virtual Enterprises. *Proc. 5th Int. Workshop on Intelligent Manufacturing Systems* (1998) 43-48
20. **CACI** *SIMPROCESS* (2000). <http://www.simprocess.com>
21. **Infocosm** *METEOR (Managing Endo-To-End OpeRations)*. Commercial site <http://infocosm.com/html/products.html>. Academic site (LSDIS, Dept. of Computer Science, Univ. of Georgia) <http://lsdis.cs.uga.edu/proj/meteor/meteor.html>
22. **Merz, M, Liberman, B and Lamersdorf, W** Using Mobile Agents to support Interorganizational Workflow Management, *Int. J. Applied Artificial Intelligence*, **11**(6) (September 1997)
23. **van der Aalst, W M P** Interorganizational Workflows – An approach based on Message Sequence Charts and Petri Nets, *Systems Analysis – Modelling – Simulation*, **34**(3) (1999) 335-367
24. **Allen, J F** Towards a General Theory of Action and Time, *Artificial Intelligence*, 23 (1984) 123-154
25. **Raposo, A B, Magalhães L P and Ricarte, I L M** *Petri Nets Based Coordination Mechanisms*. <http://www.dca.fee.unicamp.br/~alberto/pubs/IJCSSE/mechanisms/>
26. **Garbe, W** *Visual Simnet V.1.37 – Stochastic Petri-Net Simulator* (1997). <http://home.arcor-online.de/wolf.garbe/simnet.html>
27. **Starke, P H** *INA – Integrated Net Analyzer*. Institute für Informatik – Humboldt-Universität zu Berlin (1999). <http://www.informatik.hu-berlin.de/~starke/ ina.html>
28. **Raposo, A B, Magalhães L P and Ricarte, I L M** Coordinating Activities in Collaborative Environments: A High Level Petri Nets Based Approach. Accepted to the *4th World Multi-conference on Systemics, Cybernetics and Informatics*, Orlando, FL (July 2000)
29. **Nilsson, N J** *Principles of Artificial Intelligence*, Morgan Kaufmann Pubs. (1986)
30. **Magalhães L P, Raposo, A B and Ricarte, I L M** Animation Modeling with Petri Nets. *Computer & Graphics*, **22**(6) (1998) 735-743. Pergamon Press

Appendix – Other Coordination Mechanisms

In this appendix we shortly present some other coordination mechanisms of the library. The complete set of mechanisms is shown in [25].

The first mechanism presented is for dependency *Task1 before Task2* (Figure A1). The core of the model is transition *t1*, which determines the execution of Task2 and is inhibited if there is any token in *start_task1* or *P3* (Task1 is ready or being executed, respectively). The firing of *t1* sends a token to *P1*, inhibiting the firing of *t2* and, consequently, the execution of Task1. If Task2 has not been executed yet (no token in *P1*), *t2* can be fired, sending tokens to *P2*, which enables *task1*, and to *P3*, which inhibits *t1*. At the end of Task1 *t4* is fired, removing the token from *P3*. Transition *t3*

enables other executions of Task2 (after the first one, which puts a token in *P1*), which can happen independently of the presence of tokens in *start_task1* (Task1 will not happen anymore). It is important to use a timeout to ensure that Task1 returns to its initial state when it cannot be executed anymore.

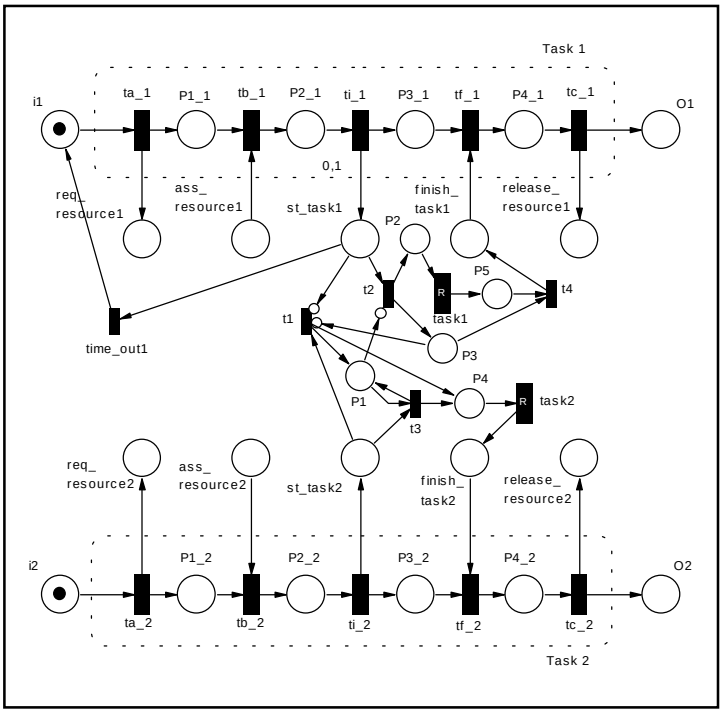

Figure A1: Coordination mechanism for *Task1 before Task2*.

Another mechanism presented here is for resource management dependency *sharing N*. The coordination mechanism for this dependency consists of a place *Pn* with N tokens representing the available resources. This place is the input place for a transition connecting *request_resource* to *assigned_resource*, defining whether there are available resources. At the end of the task, the token returns to *Pn* via *release_resource*. Figure A2 shows the model for N = 3.

Finally we show the coordination mechanism for the composite relation *simultaneity M with volatility N*. This mechanism is similar to that of *simultaneity M* (Figure 7). The difference is the inclusion of place *Pn* relative to the volatility. The timeouts presented in the figure are optional, but very important when the resource finishes. Figure A3 shows this mechanism for M = 2 and N = 4. It is important to note that N is the number of times the resource is going to be used, and not the number of groups of M tasks that is going to use it.

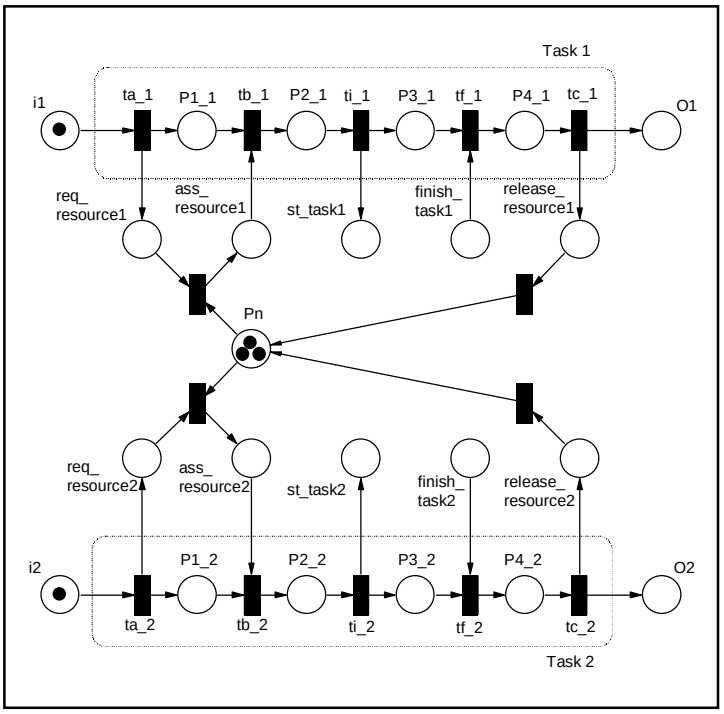

Figure A2: Resource manager for sharing N (N = 3) with two tasks able to request the resource.

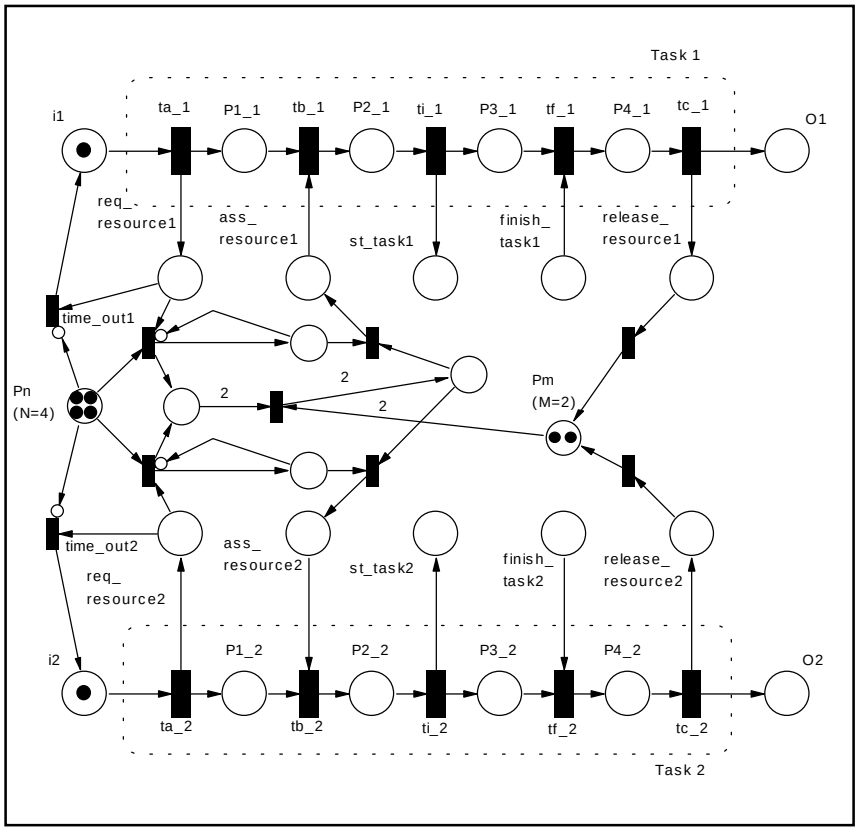

Figure A3: Resource manager for *simultaneity 2 with volatility 4*.