Defining Task Interdependencies and Coordination Mechanisms for Collaborative Systems

Alberto B. RAPOSO
abraposo@tecgraf.puc-rio.br
Computer Graphics Group (Tecgraf)

Hugo FUKS
hugo@inf.puc-rio.br
Software Engineering Laboratory (LES)

Computer Science Department – Catholic University of Rio de Janeiro (PUC-Rio)
R. Marquês de São Vicente, 225 – Rio de Janeiro, RJ, Brazil – 22453-900

Abstract. This paper addresses the issue of coordination, which is an essential matter to the specification of activities in collaborative systems. An activity can be described as a set of interdependent tasks. In order to specify task interdependencies, an extensible model encompassing a set of temporal and resource management relations is initially presented. Then, Petri Nets are used to formally model coordination mechanisms for those interdependencies. The separation between tasks and interdependencies/coordination mechanisms allows for the deployment of different coordination policies in the same collaborative system by only changing the coordination mechanisms. Moreover, the coordination mechanisms, like the related interdependencies, are generic and may be reused in several collaborative systems.

Keywords. CSCW, Coordination, Petri Nets, Synchronization, System Design.

1. Introduction

In order to work collaboratively, people need to share information (*communication*). Communication, although vital, is not enough; "it takes shared space to create shared understandings" [1]. This notion of shared workspace (including user awareness, shared objects, etc.) is called *cooperation*. To cooperate, however, people need to work harmoniously, avoiding conflicting or repetitive actions (*coordination*). These aspects (communication, cooperation and coordination) constitute a threesome frequently associated with collaboration [2], [3].

The present work treats one of those aspects – coordination, which in a broad sense may be defined as the activities responsible to ensure the effectiveness of the collaborative work. In this broad sense, coordination is a synonym of what has been called *articulation work*, defined as "a set of activities required to manage the distributed nature of cooperative work" [4]. Among the activities of articulation work, the identification of the objectives of the group work, the mapping of these objectives into tasks, the participants' selection, the distribution of tasks among them, and the coordination (in a narrow sense, as it will be seen below) of tasks execution can be mentioned.

Coordination, in a narrow definition, is "the act of managing interdependencies between activities performed to achieve a goal" [5]. In this sense, coordination is the most important

part of the articulation work because it represents the dynamic aspect of articulation, demanding renegotiation almost continuously during a collaborative effort.

Coordination, in its broad definition, is essential to any kind of collaboration. In spite of that, in its narrowest definition, coordination does not need to appear explicitly in some kinds of collaborative activities – called *loosely integrated collaborative activities* – such as those realized by means of chats or audio/videoconferences. These activities are deeply associated with social relations and generally are well coordinated by the valid "social protocol," which is characterized by the absence of any explicit coordination mechanism among the activities, trusting users' abilities to mediate interactions (the coordination is culturally established and strongly dependent on mutual awareness).

On the other hand, there is a large group of activities (*tightly integrated collaborative activities*) that require sophisticated coordination mechanisms in order to be efficiently supported by computer systems. In this kind of activity, tasks depend on one another to start, to be performed, and/or to end. Examples of tightly integrated activities may be found in workflow procedures, learningware, collaborative authoring, multi-user computer games, among others.

In this context, this paper introduces a model for the definition of generic interdependencies that occur between tasks in tightly integrated collaborative activities and proposes Petri Net-based coordination mechanisms to handle such interdependencies. The coordination model encompasses temporal and resource-related interdependencies, which have a direct mapping to the Petri Net-based coordination mechanisms.

In the sequence, several aspects related to tasks interdependencies are discussed. Section 3 introduces the coordination mechanisms for those interdependencies, also showing some examples of use. Then, in Section 4, a brief comparison with related work is presented. Finally there are conclusions and suggestions for future research.

2. The Coordination Model: Task Interdependencies

In the context of this work, a collaborative *activity* is defined as a coordinated set of tasks realized by multiple actors in order to achieve a common goal. Thus, a *task*, either atomic or expressed as a group of subtasks, is one of the building blocks of any collaborative activity. A group of subtasks could be considered to be a task when it presents no external interdependencies, that is, no interdependencies with another task that does not belong to the group. This definition of task enables the modeling of collaborative activities using several abstraction levels (see Figure 1), which facilitates the coordination specification and management.

Interdependency is a key concept in the coordination theory – if there are no dependencies between tasks to be performed in a collaborative effort, there is nothing to coordinate [6]. The approach task/interdependency or, more specifically, the clear separation between "articulation work, i.e., the work devoted to activity coordination and coordinated work, i.e., the work devoted to their articulated execution in the target domain" [7] is a step toward giving flexibility to coordination mechanisms, which is crucial to further use of this kind of mechanism.

One of the advantages of the separation task/interdependency is the possibility of altering coordination policies by simply altering the coordination mechanisms for the interdependencies, without the necessity of altering the core of the collaborative system. Additionally, interdependencies and their coordination mechanisms may be reused. It is possible to characterize different kinds of interdependencies and identify the coordination mechanisms to manage them, creating a set of interdependencies and respective

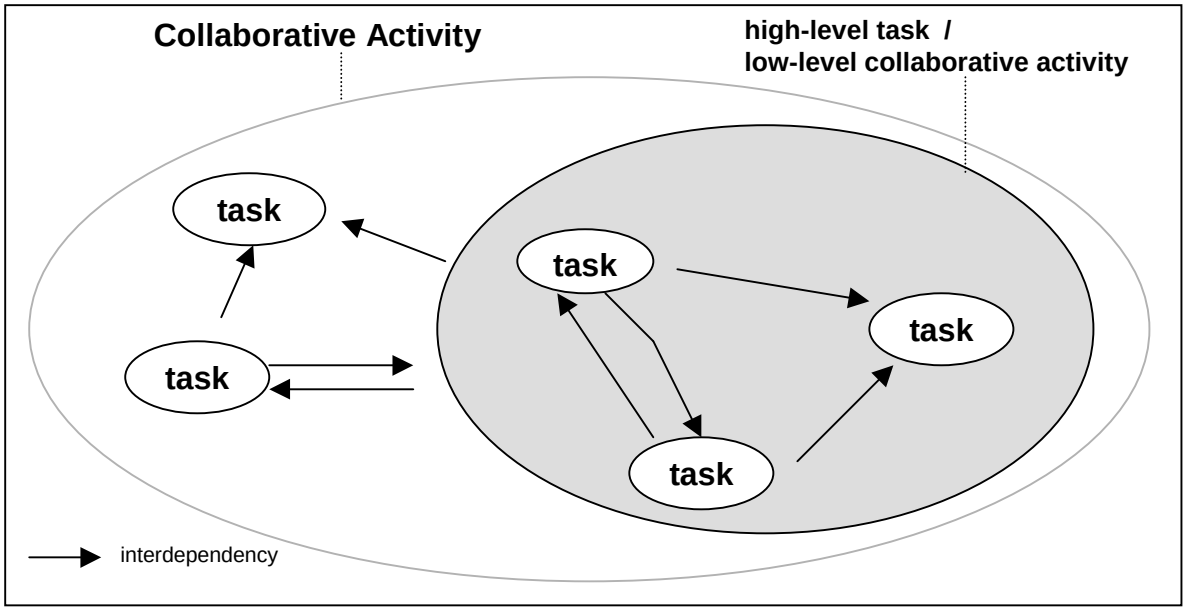

Figure 1: Hierarchical model of tasks and collaborative activities.

coordination mechanisms capable of encompassing a wide range of collaborative applications [6].

In this section, a generic set of interdependencies that occur between tasks in collaborative activities is defined. Then, in Section 3, coordination mechanisms to control those dependencies are proposed.

2.1 Basic Temporal Interdependencies

Temporal interdependencies establish the relative order of execution between a pair of tasks. The set of temporal interdependencies of the proposed model is based on temporal relations defined by J. F. Allen [8]. He proved that there is a set of primitive and mutually exclusive relations that could be applied over time intervals (i.e., any pair of time intervals are necessarily related by one and only one of Allen's relations).

A time interval is characterized by two events, which in turn are associated to time instants. The first event is the starting (initial) time of an interval *A*, denoted here i_a. The other event is the ending (final) time of the same interval, denoted f_a, always with $i_a < f_a$. According to Allen, the set of seven primitive relations shown in Figure 2 may maintain temporal information considering any pair of time intervals *A* and *B* (if one considers the inverse relations, then 13 relations can be defined, because the inverse of equals is the equals relation itself).

Based on the relations of Figure 2, a set of axioms is defined to create a temporal logic. For example, there are axioms to prove the mutual exclusion and the exhaustivity of the basic relations and others to define transitivity relations, e.g., if A during B and B before C, then it is inferred that A before C [9].

The fact of being applied over time intervals (and not over time instants) made the above relations suited for task coordination purposes, because tasks are generally non-instantaneous operations. The adaptation of Allen's primitives to the context of collaborative activities takes into account that any task *T* will take some time (from *i* to *f*) to be performed.

Nevertheless, Allen's temporal logic is defined in a context where it is essential to have properties such as the definition of a minimal set of basic relations, the mutual exclusion

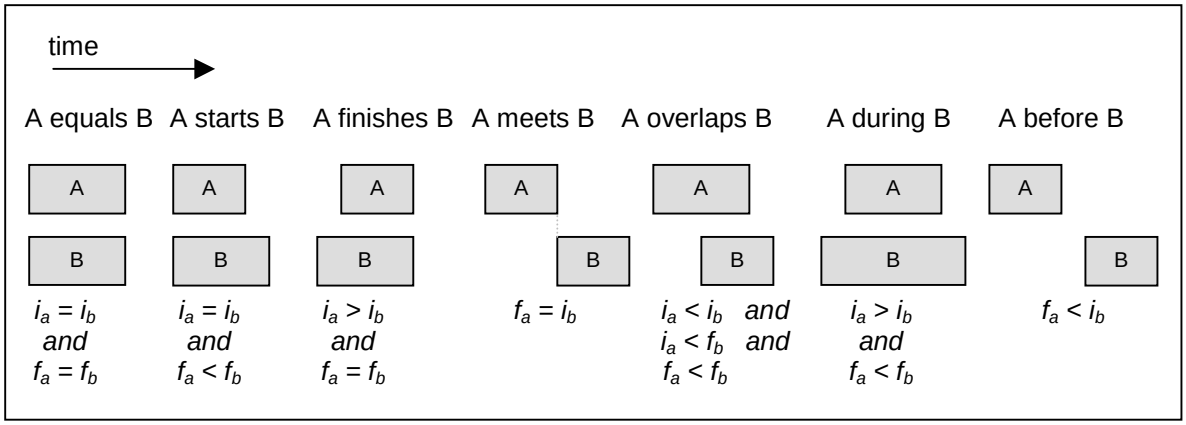

Figure 2: Allen's primitive relations between time intervals A and B.

among these relations and the possibility to make inferences over them. Temporal interdependencies between collaborative tasks, on the other hand, are inserted in a different context. What really matters here is the management of the interdependencies and the proper understanding by groupware designers.

Another drawback of Allen's relation is that they are merely descriptive, not expressing causal or functional relations between intervals [10]. For example, if tasks A and B are related by the *equals* temporal interdependency, what should the coordination mechanism do when task A is ready to begin, but not task B? Should it block the execution of task A until task B is ready, or should it force the start of task B to guarantee that the interdependency will be respected? In a different situation, if it is said that task A occurs before task B, what should be done when task B is ready but not task A? Should the coordination mechanism block task B until the end of task A, or should it allow the execution of task B, blocking future executions of task A (which would violate the relation)?

For all of these reasons, it was necessary to make some adaptations to Allen's basic relations. More than answering questions such as the ones stated above, the goal of the proposed extensions is to offer a larger set of possibilities to create coordination mechanisms that could handle many different situations. The idea is to accept Allen's seven primitives as the basic interdependencies and to provide a list of extensions, which may be viewed as specializations of them.

2.2 Active and Passive Interdependencies

The merely descriptive characteristic of Allen's temporal relations allows for different interpretations of a single interdependency. Consider, for instance, that two tasks, Ta and Tb are related by the interdependency *Ta equals Tb*. This construct establishes that the two specified tasks must be executed simultaneously. In the coordination context, this can be interpreted in two different ways. In the first sense, denoted as the active interpretation, this relation expresses that the beginning of one task should start another task; similarly, the end of one of the tasks should conclude the other task. Consider a situation in which one task is "to start a discussion session" and the other task is "to record an ongoing discussion session." From the coordination point of view, this "active equals" relationship between these two tasks would simply indicate that the second task (record the session) should follow the execution of the first task. However, a problem to proceed with the session recording would not invalidate the discussion session itself.

The second possible interpretation for any coordination mechanism is denoted as the passive interpretation. In this case, the coordination mechanism expresses a set of

conditions that should be obeyed in order to carry out the activity. Considering the same example as above, this would be the case whenever the session recording must be ready before the start of the discussion. Thus, a problem to record the session would delay the beginning of the discussion session until the problem is solved.

In order to deal both with active and passive interpretations, two operators were defined: *enables* and *forces*. The *enables* operator represents the passive interpretation, while *forces* represents the active one. These operations may be applied on the initial and final instants of each interdependent task. Additionally, these extreme points have two states, *ready* and *concluded*, indicating, respectively, that the task is ready to start (or finish) and that it has already started (or finished). These states are used in the first operand, indicating that it will enable or force the second operand before (*ready*) or after (*concluded*) its own execution.

Consider, for example, two tasks *Ta* and *Tb*, with initial and final points i_a, i_b, f_a and f_b. The interdependency *Ta equals Tb* may be extended into several interpretations.

For the simultaneous beginning:

- i_a *(ready) enables* i_b *AND* i_b *(ready) enables* i_a – this statement indicates the passive situation, in which the tasks will start their execution only when both are ready (i.e., *Tb* will be enabled to start only when *Ta* is ready to start, and vice-versa), but neither will force the execution of the other.
- i_a *(ready) forces* i_b – in this situation, when *T*a is ready to begin, *Tb* is forced to start, indicating a master/slave active interdependency (similarly, *Tb* could be considered the master if i_b *(ready) forces* i_a).
- i_a *(ready) forces* i_b *AND* i_b *(ready) forces* i_a – active interdependency with no master (the beginning of each task will force the beginning of the other).
- i_a *(ready) forces* i_b *AND* i_b *(ready) enables* i_a – *Ta* is the master, forcing the beginning of *Tb*, but *Ta* will only be started when *Tb* is ready (returning to the discussion session example, this situation indicates that the beginning of the section – *Ta* – would force the recording task, but if there is a problem with the recorder, the session will not start).

Similar interpretations are applied to the simultaneous end of both tasks:

- f_a *(ready) enables* f_b *AND* f_b *(ready) enables* f_a – passive situation. Tasks will finish their execution only when both are ready to finish, but neither will force the end of the other.
- f_a *(ready) forces* f_b – when *T*a is ready to finish, *Tb* is also forced to finish (master/slave).
- f_a *(ready) forces* f_b *AND* f_b *(ready) forces* f_a – active interdependency with no master.
- f_a *(ready) forces* f_b *AND* f_b *(ready) enables* f_a – *Ta* is the master, but has to wait until *Tb* is ready to finish.

Thus, interdependency *Ta equals Tb* may be composed of any combination of the above simultaneous beginning and end situations (for example, it could have a master/slave beginning and a passive end).

The interdependency *Ta starts Tb* has the same interpretations than *equals* for the simultaneous beginning of both tasks. Regarding the end of the tasks, the original relation imposes that *Ta* finishes before *Tb*, which is represented by f_a *(concluded) enables* f_b (the end of *Tb* will be enabled only after the end of *Ta*, as indicated by the concluded state of f_a). However, it is not always interesting to impose any restriction to the end of tasks that have started together. In this case, it is possible to relax the original relation simply not including any relation for the ends of the tasks.

For the simultaneous end of the tasks, interdependency *Ta finishes Tb* has the same interpretations as *equals*. For the beginning of the tasks, it is necessary to impose at least

the following restrictions, i_a *(concluded) enables* f_b *AND* i_b *(concluded) enables* f_a. These restrictions are necessary to guarantee that a task will only be ready to finish when the other has already started (otherwise, in the active interpretation, a task could force the end of a task that has not started). This situation does not make any assumption about which task will start before (relaxing Allen's original relation). To follow the restriction of the original relation, imposing that *Ta* starts after *Tb*, it is necessary to add the statement i_b *(concluded) enables* i_a.

The interdependency *Ta meets Tb* may have the following interpretations:

- i_b *(ready) enables* f_a *AND* f_a *(concluded) forces* i_b – passive situation. *Ta* will only be able to finish when *Tb* is ready to start, and the end of *Ta* forces the beginning of *Tb*, in order to respect the interdependency.
- f_a *(concluded) forces* i_b – active situation where *Ta* is the master. The difference with the previous situation is that the end of *Ta* does not have to wait until *Tb* is ready to begin.
- i_b *(ready) forces* f_a – active situation where *Tb* is the master. When it is ready to begin, *Ta* is forced to finish.
- f_a *(concluded) forces* i_b *AND* i_b *(ready) forces* f_a – active situation with no specific master.

The interdependency *Ta overlaps Tb* has the following statement for its passive interpretation, i_a *(concluded) enables* i_b *AND* i_b *(concluded) enables* f_a *AND* f_a *(concluded) enables* f_b. It is possible to relax the original relation removing, for example, the last part of the previous statement (in this case, it would not matter which task finishes first). An active interpretation of this interdependency could be f_a *(ready) forces* i_b, indicating that *Tb* will be forced to start when *Ta* wants to finish.

Ta during Tb may have the following interpretations:

- i_b *(concluded) enables* i_a *AND* f_a *(concluded) enables* f_b – passive situation.
- f_b *(ready) forces* i_a *AND* f_a *(concluded) enables* f_b – in this case, *Tb* is the master, forcing the execution of *Ta* before the master's end.
- i_a *(ready) forces* i_b *AND* i_b *(concluded) enables* i_a *AND* f_a *(concluded) enables* f_b – *Ta* is the master, *forcing* the start of *Tb* when the master is ready to execute.

Finally, *Ta before Tb* allows for a single passive interpretation, f_a *(concluded) enables* i_b. An active interpretation is not possible considering just the initial and final instants of both tasks, because that interdependency imposes an interval between both tasks (otherwise, it becomes the *meets* interdependency). The way to create an active interpretation for this interdependency is by defining a delay parameter for the *forces* operator, such as in the statement f_a *(concluded) forces[5s]* i_b, indicating that *Tb* will be forced to start 5 seconds after *Ta*.

The interpretations presented in this section do not claim to be exhaustive, but show the large number of possibilities that arise when considering the passive and active interpretations of temporal interdependencies.

2.3 Interdependencies Allowing a Variable Number of Executions

In spite of the operators *enables* and *forces* created to adapt Allen's relations for active and passive coordination interpretations, there are undefined situations remaining. Such a situation occurs, for example, in *Ta before Tb*. After *Ta* and *Tb* have been finished, how should the coordination mechanism proceed if *Tb* wants to start again? Should it allows its execution, since *Ta* has already been executed (one to many relationship), or should it makes *Tb* wait until *Ta* is executed again (one to one relationship)? A similar doubt arises

for *Ta during Tb*, i.e., how many times *Ta* is allowed to execute during a single execution of *Tb*?

In order to deal with such situations, it was necessary to include an optional parameter for the *enables* operator. This parameter indicates the number of times a condition (first operand) enables the event (second operand).

For example, to define that *Ta during Tb* allows the maximum of two executions of *Ta* for each execution of *Tb*, the following statements are used i_b *(concluded) enables[2]* i_a *AND* f_a *(concluded) enables* f_b. A similar situation occurs for *Ta before Tb*, as illustrated by the statement f_a *(concluded) enables[3]* i_b, indicating that, after each execution of *Ta*, *Tb* is allowed to execute up to three times. It is also possible to define that there is no restriction on the number of times a task may be executed after or during another (equivalent to define the parameter as infinite).

2.4 Blocking Interdependencies

In order to enhance the flexibility of the model, it is also necessary to create the *blocks* and *unblocks* operators that, respectively disable and re-enable the execution of an event (second operand) when the state of the first operand is reached. The use of these operators, for example, allows for a new interpretation of *Ta before Tb*:

i_b *(concluded) blocks* i_a – in this case, there is a restriction in the execution of *Ta*, which may not be executed anymore if *Tb* has already started its execution. There is no restriction on the execution of *Tb* (*Tb* does not have to wait for the execution of *Ta*, as would happen with the situation given by f_a *(concluded) enables* i_b).

The blocking situations should be carefully used, since they could create deadlocks.

2.5 Resource Management Interdependencies

According to the coordination model by Ellis and Wainer [11], there are two levels of coordination, one related to the activity level (temporal – the sequencing of tasks that make up an activity) and the other related to object level (resource – "how the system deals with multiple participants' sequential or simultaneous access to some set of objects").

Resource-related interdependencies may be represented by combinations of temporal relations. For example, if two tasks, *Ta* and *Tb*, may not use the same resource simultaneously, it is possible to define a "not parallel" dependency as the following statement, i_a *(ready) blocks* i_b *AND* f_a *(concluded) unblocks* i_b *AND* i_b *(ready) blocks* i_a *AND* f_b *(concluded) unblocks* i_a. However, besides being prone to deadlocks, this possibility ignores the notion of resource, which is quite important in the context of collaborative activities. Therefore, it is not sufficient to treat the problem of task interdependencies as a temporal logic problem. Moreover, considering resource management dependencies independently of temporal ones, a more flexible model is created, allowing the designer to deal with each kind of dependency separately.

Resource management interdependencies in the proposed model are complementary to temporal ones and may be used in parallel to them. This kind of interdependency deals with the distribution of resources among the tasks. Three basic resource management dependencies were defined elsewhere [12].

Sharing – a limited number of resources must be shared among several tasks.

Simultaneity – a resource is available only if a certain number of tasks request it simultaneously. It represents, for instance, a machine that may 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.

Each of the above interdependencies requires parameters indicating the number of resources to be shared, the number of tasks that must request a resource simultaneously and/or the number of times a resource may be used (volatility).

3. Coordination Mechanisms

Some conclusions that can be drawn from what has been presented in the previous section are *i)* a simple group of seven temporal relations is exploded into numerous coordination situations that should be correctly treated by the coordination mechanisms of a collaborative system; *ii)* although encompassed by temporal interdependencies, resource management ones also are necessary and *iii)* instead of defining coordination mechanisms for each interpretation of interdependencies, it is necessary to propose a direct mapping to construct the adequate mechanisms from the directives presented in the previous section.

In this section the last issue cited above is addressed by means of the definition of a model for the construction of coordination mechanisms based on Petri Nets (PNs). The choice for PNs as the modeling tool is justified because they are a well-established theory (there are numerous applications and techniques available) and can capture some of the main features of a collaborative environment, such as non-determinism, concurrency and synchronization of asynchronous processes. Moreover, PNs accommodate models at different abstraction levels and are amenable both to simulation and formal verification. In the following some PN fundamentals are briefly overviewed, then the coordination mechanisms are presented.

3.1 Petri Nets Fundamentals

PNs [13], [14] are a modeling tool applicable to a variety of fields and systems, specially suited for systems with concurrency, synchronization and event conflicts. 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; $F \subseteq (P \times T) \cup (T \times P)$ 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 with places and tokens, and events with transitions. A transition t is said to be enabled if each input place $P_i \in {}^\bullet t$ is marked with at least $w(P_i, t)$, which is the weight of the arc between P_i and t. It is also possible to define inhibitor arcs connecting places to transitions. In this case, the transition is enabled only if the places of origin are empty. 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 (Figure 3) which is going to 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 (inhibitor arcs are represented with a circle on the edge), with weights above. By definition, an unlabeled arc has weight 1.

In the PN of Figure 3, only transition *t2* is enabled; *t1* is not enabled because it would require two tokens in *P1* to fire, since *w(P1, t1) = 2*; *t3* is not enabled because of the

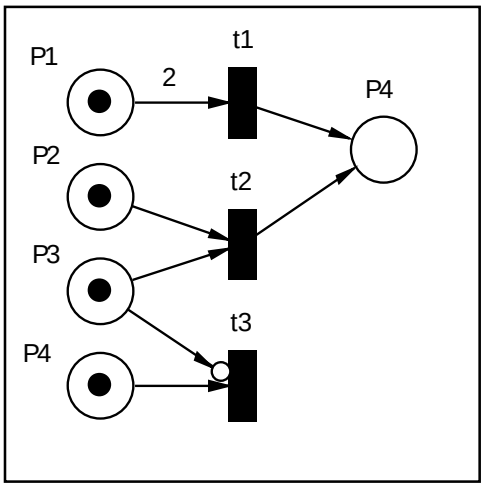

Figure 3: Example of the graphical notation of PNs.

inhibitor arc from *P3* (this place would have to be empty to enable *t3*). 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.

3.2 Petri Nets-Based Coordination Mechanisms

In the proposed scheme, the design of a collaborative environment is divided into three distinct hierarchical levels, *workflow*, *coordination* and *execution* (Figure 4). In the *workflow* level, each participant's behavior is modeled separately, establishing the interdependencies between tasks of the same participant or those of different ones. The *coordination* level is built under the workflow level by the expansion of interdependent tasks according to a PN-based model and the insertion of correspondent coordination mechanisms between them. The environment model is simulated and analyzed at this level. The *execution* level deals with the actual execution of tasks in the system.

During the passage from the workflow to the coordination level, each task that has an interdependency with another is expanded in the subnet presented in Figure 5. In this model, events *i* and *f* (start and end of the task) are represented as transitions, while states *ready* and *concluded* are represented as places connected to the respective transitions. After the firing of *i*, the flow is divided into two parallel paths, one indicating that the task is in execution – *i(concluded)* – and another representing the interaction with the task's execution in the system. The task execution is modeled by means of a transition with token reservation (represented with the letter "R"), which is a non-instantaneous transition – tokens are removed from its input places when it fires and only some time later are added to its output places, representing the duration of the task.

When considering two tasks related by interdependencies, it is necessary to correctly interconnect places and transitions of both models for creating the respective coordination mechanisms. In order to do that, it is necessary to define how to map the operators and parameters previously defined to the PN models.

The model of the *enables* operator is quite simple. It is modeled by an arc from the place representing the first operand to the transition representing the second one. For example, i_a *(concluded) enables* i_b is represented by an arc from place i_a *(concluded)* to transition i_b. It is also necessary to add 1 to the weight of the arc arriving at the first operand, because this allows that place to enable both the normal flow of the task and the event given by the second operand. To illustrate that, consider the interdependency *Ta equals Tb* in the passive interpretation, i.e., i_a *(ready) enables* i_b *AND* i_b *(ready) enables* i_a *AND* f_a *(ready) enables* f_b

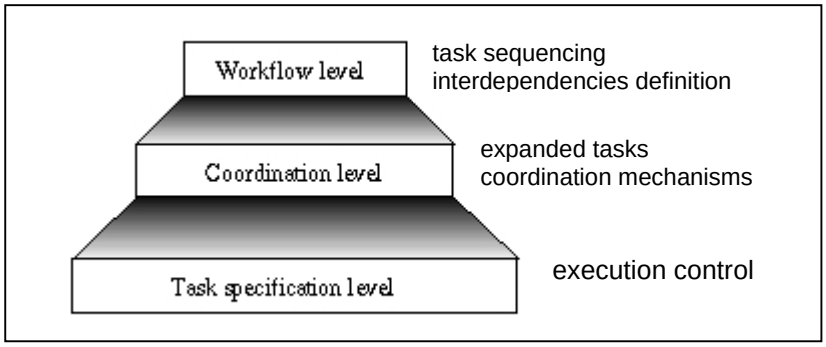

Figure 4: Coordination mechanisms design levels.

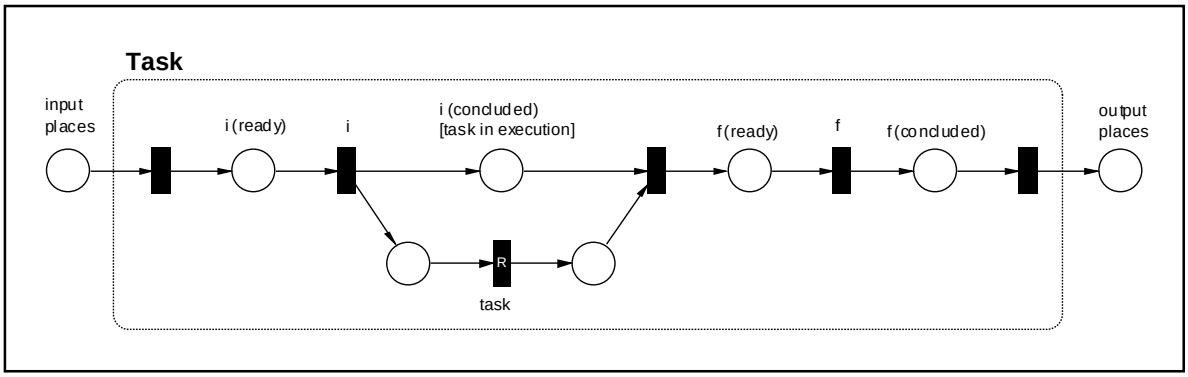

Figure 5: PN representation of an interdependent task at the coordination level.

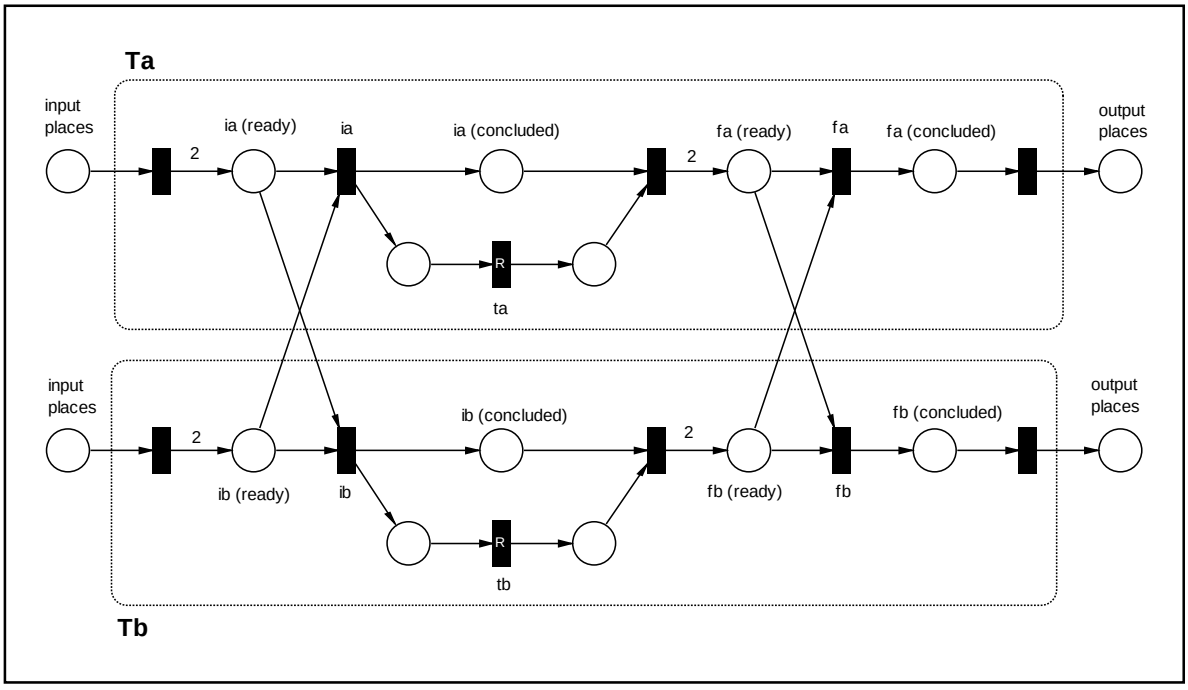

Figure 6: Coordination mechanism for *Ta equals Tb*, passive interpretation.

AND f_b *(ready) enables* f_a. The coordination mechanism for this interdependency is illustrated in Figure 6.

The *enables* operator also requires a parameter when it is necessary to enable a variable number of executions of the second task (as presented in Section 2.3). In this case, instead of adding 1 to the weight of the arc arriving at the first operand, it should be added the number given by the parameter. If that number is infinite, it is necessary to include a return

arc from the transition representing the second operand to the place representing the first operand.

The *forces* operator requires an additional transition in the coordination mechanism. If the forced event is ready, then *forces* functions exactly like *enables*. However, if the forced event is not ready, it is necessary to include an alternative transition connecting the forcing state (first operand) to the output places of the second operand, indicating, in the PN model, that it has been forced to happen. Figure 7 illustrates the interdependency *Ta meets Tb* in the active situation when *Ta* is the master, given by f_a *(concluded) forces* i_b. In this mechanism, similar to that of *enables*, there is an arc from f_a *(concluded)* to i_b and an additional weight on the arc arriving at f_a *(concluded)*. Additionally, the alternative transition appears between f_a *(concluded)* and the output places of i_b. The alternative transition must have lower priority than the others in order to avoid that it be fired when the forced event is ready.

In certain situations, it is necessary to include a delay parameter in the *forces* operator (Section 2.2). In these cases, it is possible to use timed transitions, which are fired only a determined time after enabled.

The *blocks* operator is constructed exactly like *enables*, but using an inhibitor arc instead of a normal one. In this case, the block is permanent, because the extra token added to the blocking place is not removed. The *unblocks* operator is modeled by a transition that removes that extra token.

Regarding resource management interdependencies, it is necessary to include a place whose tokens indicate the available resources. For example, Figure 8 represents the situation where two tasks share a single resource, represented as a token in place *R*. The start of each task depends on the availability of that resource. The end of the task releases the resource, if it is not volatile.

3.3 Example

In order to get a general idea of the deployment of the proposed coordination model, a collaborative authoring activity is going to be modeled. Suppose that in this scenario there are two participants, the author (user A) and the reviewer (user B). The revision process should start after the beginning of the writing and may occur partially in parallel with it. Moreover, user B may alter the document to make corrections during the revision, but this may not conflict with the writing task of user A.

The workflow level of the coordination model (see Figure 4) is directly obtained from the collaborative activity description. At this level, it is necessary to describe each participant's tasks and the interdependencies between them. In the scenario described, user A has the task of writing the document (*writeA*). User B has two tasks, the revision (*reviewB*) and the alterations (*alterB*). The interdependencies between these tasks are:

writeA overlaps reviewB
alterB during reviewB
writeA and alterB sharing 1

The third interdependency above guarantees that the document will not be edited simultaneously by both users.

In order to pass to the coordination level, it is initially necessary to choose which interpretations of the interdependencies are more appropriate in this situation. For the *overlaps* interdependency, it is not necessary that the writing task forces the revision, although it may not finish before starting the revision. In this situation, the passive

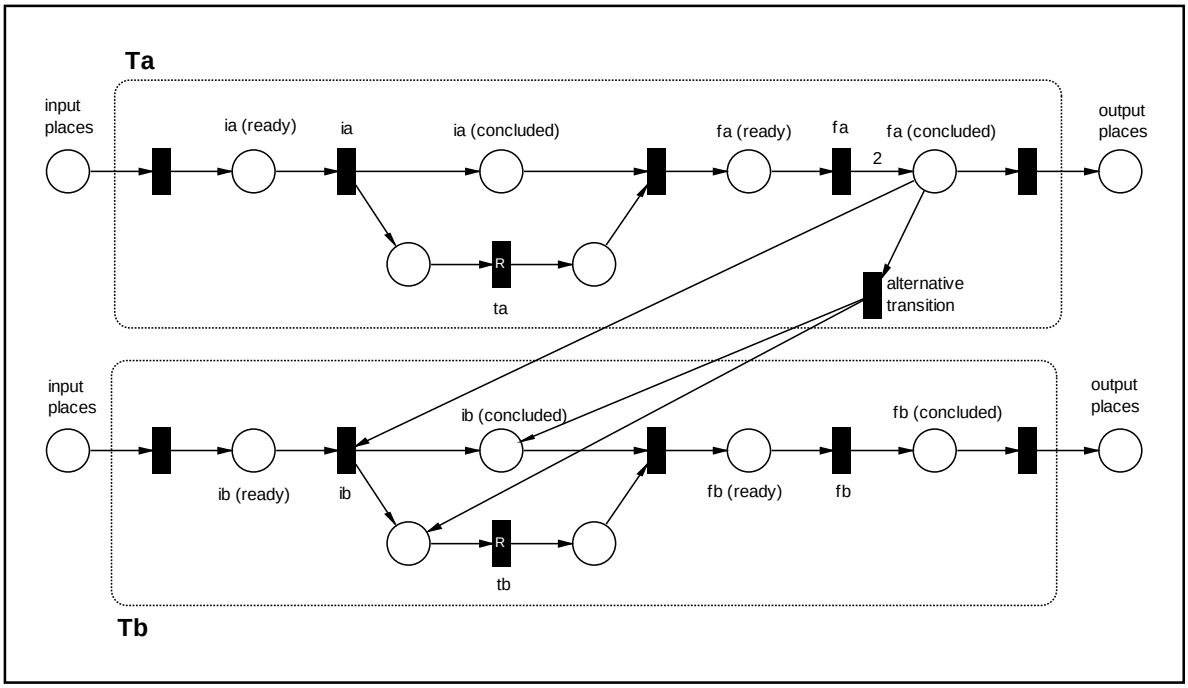

Figure 7: Coordination mechanism for *Ta meets Tb*, active interpretation – *Ta* master.

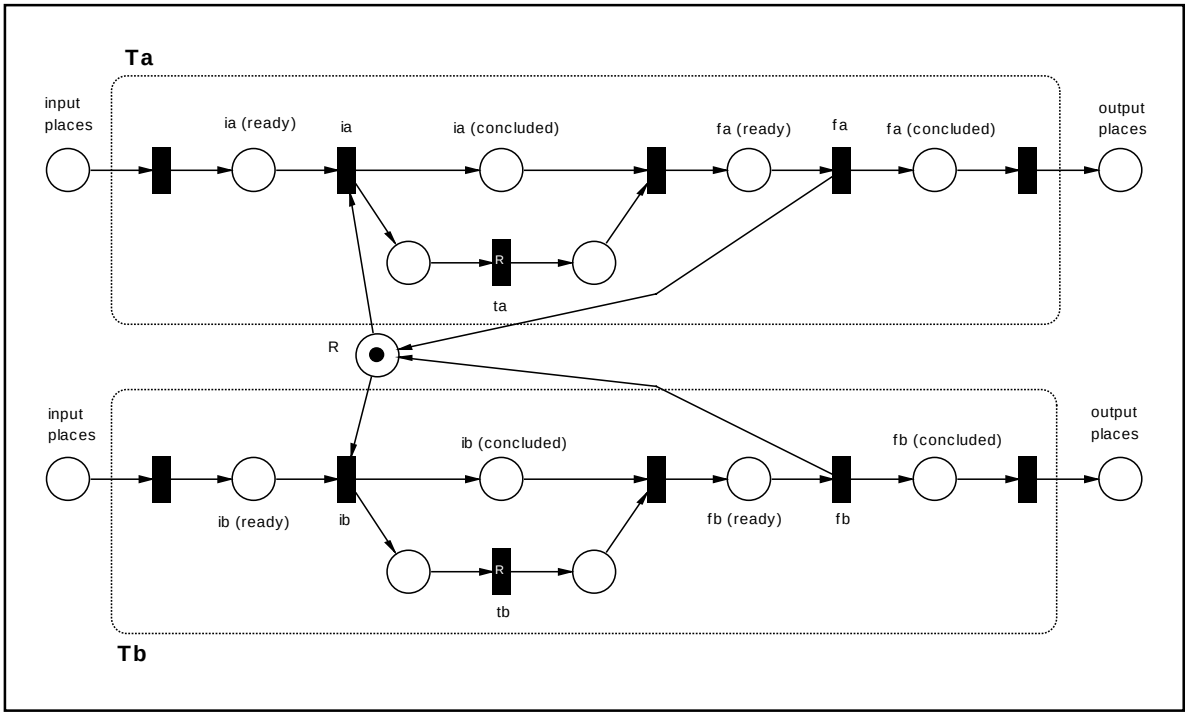

Figure 8: Coordination mechanism for two tasks sharing a single resource.

interpretation is adequate: i_{writeA} *(concluded) enables* $i_{reviewB}$ *AND* $i_{reviewB}$ *(concluded) enables* f_{writeA} *AND* f_{writeA} *(concluded) enables* $f_{reviewB}$.

For the *during* interdependency, in this case, it is enough to say that $i_{reviewB}$ *(concluded) enables* i_{alterB} *AND* $f_{reviewB}$ *(concluded) blocks* i_{alterB}.

The definitions above allow the construction of the PN-based coordination mechanisms for this example (shown in Figure 9). This kind of model, in more complex scenarios, may be very important to detect unexpected situations in the collaborative activity. Furthermore, it could constitute the basis for the development of coordination mechanisms at the

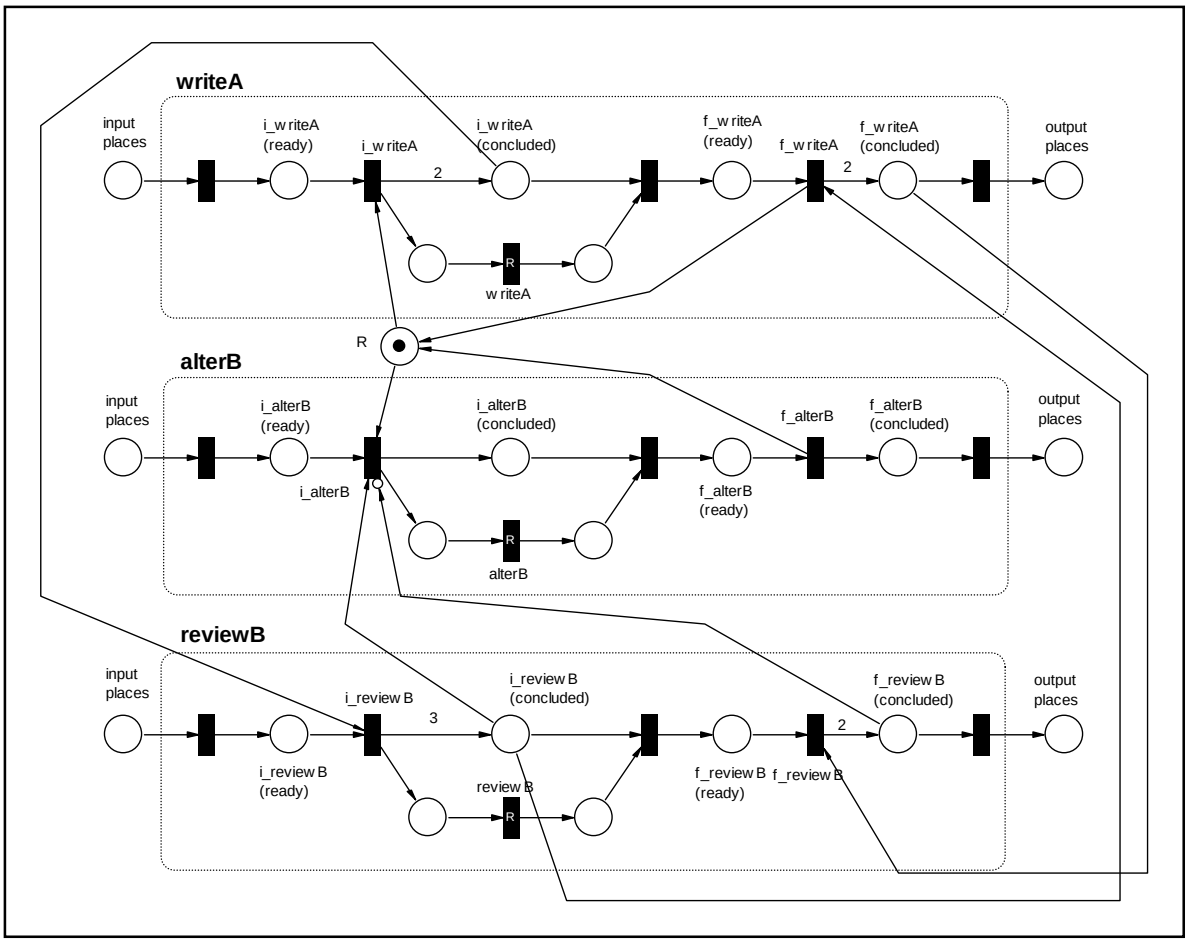

Figure 9: PN coordination mechanism for the collaborative authoring example.

specification level, which interacts directly with the "real tasks" in the system. An implementation of coordination mechanisms at this level using software components that follows a PN model was presented elsewhere [15].

4. Related Work

In order to give support to tightly integrated collaborative activities, many coordination mechanisms have been proposed in the context of some collaborative systems. The first generation of coordination models, proposed in the mid-1980s, was restricted to specific scenarios, with rigidly defined protocols (e.g., [16], [17]). Eventually, there would be situations not predicted by the specified protocols, restraining the application of the defined mechanisms. Therefore, more recent models strive for flexibility, with coordination mechanisms that can be adapted for each application needs. The so-called second generation of coordination models looks for the development of systems with at least one of the following three characteristics, which are *accessibility*, *interoperability*, and *flexibility*.

Accessibility is related to exposing the coordination mechanisms to system users rather than having them deeply embedded in the system implementation. In this case, users may be either system designers or end users. Examples of such a line of development include the Oval tool [18] and the ABACO/Ariadne [19].

Interoperability is an essential concept in the field of information systems integration, spanning from infrastructure services to business processes [20]. Originally concerned with the distributed and heterogeneous nature of modern systems, interoperability brings new concerns to coordination. Besides the obvious need to control activities defined within a

group (or intra-group coordination), the need for inter-group coordination may arise, a problem similar in nature to the integration of inter-organizational processes. Interoperability has been the focus of systems such as Reconciler [7], which aims to manage groups at the semantic level through the conciliation of conflicts.

Flexibility focuses on the possibility of dynamically allowing redefinition and temporary modifications in the coordination scheme. An example of a system that incorporates the notion of flexible coordination mechanisms is Intermezzo [21]. In this system, data objects with controlled access are assigned to groups of users with a certain role, and these roles may change during the collaboration.

The coordination model presented in this work may be fitted within the second generation because it offers a degree of flexibility through separation of tasks and interdependencies (facilitating the changes of coordination policies) and is adequate for dealing with some interoperability aspects. This is so in the sense that the interdependencies are generic (i.e., may be applied to a wide range of collaborative applications) and the implementation of coordination mechanisms may be realized by any tool. Although the use of PN-based coordination mechanisms has been stressed, the model clearly separates interdependencies from their coordination mechanisms, enabling the use of different implementation tools for the coordination mechanisms. Regarding accessibility, the proposed model exposes the coordination mechanisms to system designers, but not to the users.

The idea of creating a set of task interdependencies and respective coordination mechanisms was proposed in the coordination theory of T. Malone and K. Crowston. They defined three types of elementary resource-based dependencies (*flow*, *fit* and *sharing*) and worked with the hypothesis that all other dependencies could be defined as combinations or specializations of these basic types [22]. The coordination theory was the inspiration of the genres coordination proposal, which stresses the coordination in relation to resources, place and time [23]. Another work that should be mentioned uses the interdependencies among activities to workflow management [24]. In this case, interdependencies are defined as "constraints on the occurrence and temporal order of events", and are controlled by coordination mechanisms defined as finite state automata, which guarantee that they are not violated. Some of these ideas originated the work presented here, which is refined by defining a larger set of basic interdependencies and modeling their respective coordination mechanisms.

5. Conclusion

The necessity of coordination mechanisms to regulate interactions in collaborative systems has been the center of a heated discussion (e.g., [25], [26]). In spite of that, there is a recent trend to conciliate these ideas, trying to bridge the gap between coordination approaches for loosely and tightly integrated collaborative activities, arguing that both approaches are "seamlessly meshed and blended in the course of real world cooperative activities" [27]. In this context, this paper introduced an approach for the coordination of tightly integrated collaborative activities.

In the proposed approach, a basic set of interdependencies was defined to encompass a large number of situations, including both temporal and resource management dependencies. The mapping from these interdependencies to PN-based coordination mechanisms was also shown. By means of the PN model, the collaborative environment may be simulated and analyzed, enabling the anticipation of possible problems. Moreover, considering that tightly coupled collaborative activities may be decomposed into a set of

interdependent tasks, such coordination mechanisms provide a framework for the design of collaborative systems.

A continuation of this work is the further development of software components to implement the coordination mechanisms – an initial approach has been shown elsewhere [15]. The component model will standardize an event-based interaction between tasks and associated coordination mechanisms in an implementation independent manner.

Another possibility to be explored is the use of fuzzy coordination mechanisms, which may bring a higher degree of flexibility and manageability to collaborative systems. Conventional temporal interdependencies do not accept scenarios such as starting a task execution when another one is "almost finishing," because tasks may only be synchronized by their starting or finishing instants. Such modeling imprecision is important because it offers application designers a higher degree of flexibility to focus on their customized version of the interdependency in a manner more closely related to subjective human reasoning. The fuzzy sets theory offers adequate resources to implement coordination mechanisms with such degree of imprecision [28].

Acknowledgements. Part of this work was realized while the first author was sponsored by FAPESP (Foundation for Research Support of the State of São Paulo), working as a post-doctoral fellow at the Dept. of Computer Engineering and Industrial Automation, School of Electrical and Computer Engineering, State University of Campinas, Brazil. The second author has a researcher productivity allowance from CNPq (Brazilian National Research Council), grant n. 524557/96-9. Thanks also to Profs. Léo Magalhães and Ivan Ricarte for their helpful insights.

References

[1] Schrage, M. (1995): *No more Teams! Mastering the Dynamics of Creative Collaboration*. New York: Currency Doubleday

[2] Ellis, C. A., S. J. Gibbs and G. L. Rein (1991): Groupware: Some Issues and Experiences. *Communications of the ACM*, vol. 34, no. 1, pp. 38-58

[3] Fuks, H., C. Laufer, R. Cohen and M. Blois (1999): Communication, Coordination and Cooperation in Distance Education. In *AMCIS'99. Proceedings of V Americas Conference on Information Systems, Milwaukee, USA, August 13-15, 1999*. Association for Information Systems (AIS), pp. 130-132

[4] Schmidt, K. and L. J. Bannon (1992): Taking CSCW Seriously – Supporting Articulation Work. *Computer Supported Cooperative Work (CSCW) – An International Journal*, vol. 1, nos. 1-2, pp. 7-40

[5] Malone, T. W. and K. Crowston (1990): What is Coordination Theory and How Can It Help Design Cooperative Work Systems? In *CSCW'90. Proceedings of the Conference on Computer Supported Cooperative Work, Los Angeles, USA, October 7-10, 1990*. New York: ACM Press, pp. 357-370

[6] Malone, T. W. and K. Crowston (1994): The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, vol. 26, no. 1, pp. 87-119

[7] Simone, C., G. Mark and D. Giubbilei (1999): Interoperability as a Means of Articulation Work. In *WACC'99. Proceedings of the International Joint Conference on Work Activities Coordination and Collaboration, San Francisco, California, February 22-25, 1999*. New York: ACM Press, pp. 39-48

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

[9] Allen, J. F. (1983): Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, vol. 26, no. 11, pp. 832-843

[10] Duda, A. and C. Keramane (1995): Structured Temporal Composition of Multimedia Data. In *IW-MMDBMS. Proceedings of the International Workshop on Multi-Media Database Management Systems, Blue Mountain Lake, USA, August 28-30, 1995*. Los Alamitos, CA: IEEE Computer Society Press, pp. 136-142

[11] Ellis, C. A. and J. Wainer (1994): A Conceptual Model of Groupware. In *CSCW'94. Proceedings of the Conference on Computer Supported Cooperative Work, Chapel Hill, USA, October 22-26, 1994*. New York: ACM Press, pp. 79-88

[12] Raposo, A. B., L. P. Magalhães and I. L. M. Ricarte (2000): Petri Nets Based Coordination Mechanisms for Multi-Workflow Environments. *International Journal of Computer Systems Science & Engineering*, vol. 15, no. 5, pp. 315-326

[13] Petri, C. A. (1962): Kommunikation mit Automaten. *Schriften des IIM Nr. 3*. Bonn: Institute für Instrumentelle Mathematik

[14] Murata, T. (1989): Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE* , vol. 77, no. 4, pp. 541–580

[15] Raposo, A. B., A. J. A. da Cruz, C. M. Adriano and L. P. Magalhães (2001a): Coordination Components for Collaborative Virtual Environments. *Computers & Graphics*, vol. 25, no. 6, pp. 1025-1039

[16] Flores, F., M. Graves, B. Hartfield and T. Winograd (1988): Computer Systems and the Design of Organizational Interaction. *ACM Transactions on Office Information Systems*, vol. 6, no. 2, pp. 153-172

[17] Laufer, C. C. and H. Fuks (1995): ACCORD: Conversation Cliches for Cooperation. In *COOP'95. Proceedings of the First International Workshop on the Design of Cooperative Systems, Antibes-Juan-les-Pins, France, January 25-27, 1995*. Rocquencourt: INRIA Press, pp. 351-369

[18] Malone, T. W., K.-W. Lai and C. Fry (1995): Experiments with Oval: A Radically Tailorable Tool for Cooperative Work. *ACM Transactions on Information Systems*, vol. 13, no. 2, pp. 177-205

[19] Schmidt, K. and C. Simone (1996): Coordination mechanisms: Towards a conceptual foundation of CSCW systems design. *Computer Supported Cooperative Work (CSCW) – The Journal of Collaborative Computing*, vol. 5, nos. 2-3, pp. 155-200

[20] Hasselbring W. (2000): Information System Integration. *Communications of the ACM*, vol. 43, no. 6, pp. 33-38

[21] Edwards, W. K. (1996): Policies and Roles in Collaborative Applications. In *CSCW'96. Proceedings of the Conference on Computer Supported Cooperative Work, Boston, USA, November 16-20, 1996*. New York: ACM Press, pp. 11-20

[22] Malone, T. W. et al. (1999): Tools for inventing organizations: Toward a handbook of organizational process. *Management Science*, vol. 45, pp. 425-443.

[23] Yoshioka, T. and G. Herman (2000). *Coordinating Information Using Genres*. Center for Coordination Science, Sloan School of Management, MIT, Working Paper CCS WP#214.

[24] Attie, P. C., M. P. Singh, E. Emerson, A. Sheth and M. Rusinkiewicz (1996): Scheduling workflows by enforcing intertask dependencies. *Distributed Systems Engineering Journal*, vol. 3, no. 4, pp. 222-238.

[25] Suchman, L. A. (1994): Do Categories Have Politics? *Computer Supported Cooperative Work (CSCW) – An International Journal*, vol. 2, no. 3, pp. 177-190

[26] Winograd, T. (1994): Categories, Disciplines, and Social Coordination. *Computer Supported Cooperative Work (CSCW) – An International Journal*, vol. 2, no. 3, pp. 191-197

[27] Schmidt, K. and C. Simone (2000): Mind the gap! Towards a unified view of CSCW. In *COOP 2000. Proceedings of the 4th International Conference on the Design of Cooperative Systems, Sophia Antipolis, France, May 23-26, 2000*

[28] Raposo, A. B., A. L. V. Coelho, L. P. Magalhães and I. L. M. Ricarte (2001b): Using Fuzzy Petri Nets to Coordinate Collaborative Activities. In: *Proceedings of the Joint 9th IFSA (International Fuzzy Systems Association) World Congress and 20th NAFIPS (North American Fuzzy Information Processing Society) International Conference, Vancouver, Canada, July 25-28, 2001*. Piscataway, NJ: IEEE, pp. 1494-1499