Proc. of the 7th Workshop on Groupware - CRIWG'2001, p.170 - 179. Darmstadt, Germany. IEEE Computer Society Press, 2001.

Coordination of Collaborative Activities: A Framework for the Definition of Tasks Interdependencies

Alberto B. Raposo, Léo P. Magalhães, Ivan L. M. Ricarte
Dept. of Computer Engineering and Industrial Automation
School of Electrical and Computer Engineering
State University of Campinas (UNICAMP) – Brazil
{alberto, leopini, ricarte}@dca.fee.unicamp.br

Hugo Fuks
Software Engineering Laboratory
Computer Science Dept.
Catholic Univ. of Rio de Janeiro –Brazil
hugo@inf.puc-rio.br

Abstract

The coordination of interdependencies between tasks in collaborative environments is a very important and difficult endeavor. The separation between tasks and interdependencies allows for the use of different coordination policies in the same collaborative environment by changing only the coordination mechanisms that control the interdependencies. This paper presents a framework for the definition of interdependencies that frequently occur in collaborative activities. By means of a clear characterization of interdependencies, it is possible to identify coordination mechanisms to manage them, opening the way toward a powerful coordination tool capable of encompassing a wide range of collaborative applications. An implementation of the coordination model of a collaborative virtual environment based on the proposed framework is given as example.

1. Introduction

A definition for collaborative work that is frequently cited in the literature [21] is that of Karl Marx, written in 1867. According to Marx, collaborative work is defined as "multiple individuals working together in a planned way in the same production process or in different but connected production processes." In the kernel of this definition is the notion of planning, which ensures that the collective activity results from individual tasks.

The notion of planning is realized in CSCW (Computer Supported Cooperative Work) by what has been called *articulation work*, defined as "a set of activities required to manage the distributed nature of cooperative work" [21]. The articulation work is the additional effort required to obtain the actual collaboration from the sum of individual tasks. Among the activities of articulation work that can be mentioned are 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 of tasks execution.

An important aspect of collaborative work is the notion of task interdependency. This interdependency is always positive, in the sense that each participant wants the works of others to succeed. However, it is not always harmonious. It is necessary for coordination between tasks to exist in order to guarantee the efficiency of the collaboration. Without coordination, there is the risk that participants would get involved in conflicting or repetitive tasks. Coordination, in this context, is defined as "the act of managing interdependencies between activities performed to achieve a goal" [11]. It is the most important of the articulation work's activities because it represents the dynamic aspect of articulation, demanding to be "renegotiated" almost continuously during a collaborative effort. The other activities of the articulation work are concluded before the beginning of the collaboration and seldom need to be altered.

The great challenge in proposing coordination mechanisms to control collaborative activities is to achieve the flexibility demanded by the dynamism of the interaction between partners. A step toward this flexibility is achieved by means of a 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" [23].

One of the advantages of this approach 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 coordination mechanisms capable of encompassing a wide range of collaborative applications [12].

In this context, the present paper introduces a framework for the definition of generic interdependencies that occur between tasks in collaborative activities. In the following section related works are discussed. The interdependencies are studied in Section 3, where the framework is presented. Section 4 exemplifies the proposed model by means of the implementation of a collaborative virtual environment. The paper resumes with conclusions and suggestions for future research.

2. Related work

Collaborative systems with some kinds of coordination mechanisms only started to appear in the second half of the 1980s. However, they were restricted to specific scenarios, because coordination protocols were rigidly defined, restraining dynamic modifications. Collaborative systems containing such characteristics have what has been called *coordination models of first generation*. A coordination protocol, however, may not encompass all possible situations. Therefore, the user will eventually be in a situation that requires a deviation from the protocol. For that reason, in the 1990s systems were constructed with more flexible and easily modifiable coordination mechanisms (*coordination models of second generation*).

The second generation of coordination models has three main characteristics: *accessibility* to application designers and end users, *interoperability* and *flexibility*. In general, it is possible to say that, in regard to coordination, current collaborative systems are developed to achieve at least one of these functionalities.

Oval (Objects, Views, Agents and Links) is an example of a tool for the construction of collaborative applications in which the concerns about the *accessibility* of the coordination mechanisms to end users prevail [13]. Another system sharing the same concerns is Ariadne, which is a generic notation aiming to facilitate the construction of coordination mechanisms in any application involving collaborative work and, at the same time, enabling application designers and users to build and alter these mechanisms [22].

Interoperability is a generic term encompassing from the integration of the heterogeneous communication infrastructures of different organizations to the integration of business processes [10]. From the coordination point of view, interoperability imposes an even more challenging problem, because it deals with the support for inter-group collaboration, and not only intra-group (i.e., inside a well-established group, with some predefined conventions). Reconciler, for example, is a system whose main objective is to manage the interoperability between groups at the semantic level, conciliating their visions by means of the treatment of terminology and unity conflicts, among others [23].

The evidence that collaborative systems should not impose rigid work or communication patterns led to the development of systems that allow dynamic redefinitions and temporary modifications in the coordination model (*flexibility* in coordination). Intermezzo is an example of a system with these characteristics that uses the notions of policies and roles for the coordination of collaborative activities [6]. Coordination policies are defined by means of access control to data objects, which are assigned to group of users with a certain role. Roles are assigned not only before the collaboration, but they also change dynamically during the collaboration.

Coordination languages, initially devoted to the implementation of parallel applications [9], have also been developed for the construction of collaborative applications. The idea behind these languages is that it is possible to construct a complete programming model by separating the computation model from the coordination model. The computation model is accomplished by conventional programming languages and allows for the construction of isolated activities. The coordination model is what connects these isolated activities, establishing the threads execution control and the communication among them. DCWPL (Describing Collaborative Work Programming Language) is an example of coordination language for the implementation of collaborative applications [5]. This language separates the description of computational entities from the interaction rules followed by workgroups. These rules may be altered at runtime. In the present work, a similar separation is made between tasks (computation) and interdependencies (coordination).

2.1. Task interdependencies

The idea of creating a set of tasks interdependencies and respective coordination mechanisms was proposed in the coordination theory of Malone and Crowston [11]. 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 [14]. A *flow* dependency occurs when a task produces resource(s) that will be used by another task. A *fit* dependency arises when two or more tasks collectively produce the same resource. A *sharing* dependency occurs when two or more tasks use the same resource. In addition to these three elementary types, three subtypes were defined, namely, *prerequisite*, *accessibility* and *usability*. *Prerequisite* is related to time and means that a

resource must be produced before it may be used. *Accessibility* is related to place and implies having the resource in the right place for use. *Usability* is related to the resource itself, meaning that it should be "usable" for the task that needs it. Summarizing, an "available resource" means "the right thing, in the right place, at the right time."

The coordination theory was the inspiration of the genres coordination proposal, which stresses the coordination in relation to resources, place and time [25]. Resources, in particular, are treated according to three characteristics, *divisibility*, *concurrency* and *reusability*. *Divisibility* is related to how a resource is divided into smaller parts without loosing its usability. *Concurrency* refers to the simultaneous use of the complete resource (i.e., without being divided). For example, if two users want to eat the same chocolate, it should be divided; if they want to view the same Web page, it is a case of concurrency. The third characteristic of the resource, *reusability*, means that the same resource may be available several times (e.g., the Web page, but not the chocolate). Aspects related to time and place are very similar to that of the original coordination theory ("in the right place, at the right time").

Another work that should be mentioned uses the interdependencies among activities to workflow management [3]. 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. The objective is to create a global scheduler that satisfies all dependencies defined for the workflow. A limitation of this work is that it is restricted to temporal interdependencies and is specific to workflow applications.

The present work starts with some of the ideas of these previous works, and is refined by defining a larger set of basic interdependencies and new dimensions for them.

3. Task interdependencies: A framework

Before presenting the framework, it is necessary to clarify the definition of task as used in this work. In this context, tasks are the "building blocks" of a collaborative activity, which is defined as a coordinated set of tasks realized by multiple actors to achieve a common goal. Tasks may be atomic or composed of subtasks and are connected to one another through interdependencies. The granularity of a task is defined by the interdependencies it has with other tasks. A group of subtasks with no external interdependencies (i.e., interdependencies with another task that does not belong to this group) can be considered a task. For example, in the collaborative activity of writing a book by several authors, the writing of a chapter may be considered a high level task if it is a book of readings. In this case, the subtasks associated to the writing of a chapter have no relationship to those associated to another chapter. On the other hand, if it is a "regular" book (i.e., not divided into authored chapters), it is possible that only more granular operations (such as the writing of a section or a paragraph) may be considered tasks, depending on the authors' method of work.

Using this definition of task, it is possible to model a collaborative activity in several abstraction levels, which improves both the understandability and the feasibility of the interacting rules that characterizes the whole process. Collaborative activities, therefore, assume a more manageable perspective, facilitating the identification of coordination mechanisms for the interdependencies. The term "coordination mechanism" means a "specialized software device, which interacts with a specific software application so as to support articulation work with respect to the field of work as represented by the data structures and functionalities of that application" [22].

The proposed framework is composed of two parts. Initially (Section 3.1) a set of basic interdependencies that occurs among tasks is defined. Then, some dimensions for the interdependencies are defined (Section 3.2), which may be viewed as additional characteristics or specializations of them that will guide the identification of adequate coordination mechanisms.

3.1. Basic interdependencies

Interdependencies in the proposed framework are divided into two types, *temporal* and *resource management*. Temporal interdependencies establish the execution order of tasks. Resource management dependencies are complementary to and independent of temporal ones and deal with resource distribution among tasks.

This separation between temporal and resource management dependencies agrees with the coordination model proposed by Ellis and Wainer [7]. According to this model, the coordination in collaborative systems could occur on two levels, the activity level and the object level. At the activity level, the coordination model refers to temporal dependencies, describing "the sequencing of activities [tasks] that make up a procedure [collaborative activity]." At the object level, the coordination model refers to resource management dependencies, describing "how the system deals with multiple participants' sequential or simultaneous access to some set of objects."

3.1.1. Temporal interdependencies. Temporal dependencies establish an execution order for the tasks. An example of such kind of dependency occurs in e-commerce, in which a consumer order may only be canceled before the product is delivered [17].

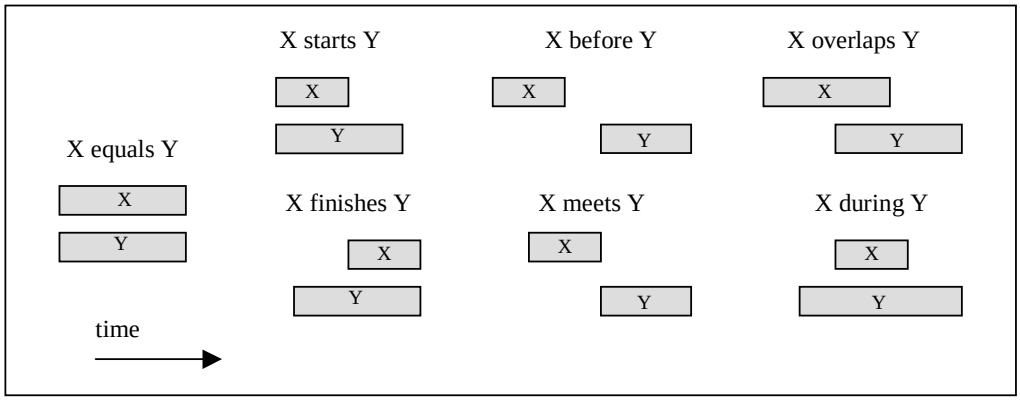

Figure 1. Allen's primitive relations between time intervals X and Y.

The set of temporal interdependencies of the framework is based on temporal relations defined by Allen [2]. According to him, there is a set of primitive and mutually exclusive relations that could be applied over time intervals (and not over time instants). This characteristic made these relations suited for task coordination purposes, because tasks are generally non-instantaneous operations. Allen's interval algebra is based on seven basic relations, as illustrated in Figure 1.

Based on the relations of Figure 1, a set of axioms is defined to create the 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 X during Y and Y before Z, then it is inferred that X before Z) [1].

The temporal logic of Allen 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 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. For this reason, it was necessary to make some adaptations to Allen's basic relations, adding a couple of new relations and creating some variations of those originally proposed. The main difference in the context of collaborative activities is that it is possible to relax some restrictions imposed by the original relations. This introduces a degree of redundancy from temporal logic's point of view, but makes the coordination model more understandable and manageable. The result of the adaptation of Allen's relations for the context of collaborative activities is the set of 13 temporal interdependencies presented below.

Consider two tasks T1 and T2 that occur, respectively, in time intervals $[t1_i, t1_f)$ and $[t2_i, t2_f)$.

T1 equals T2 ($t1_i = t2_i$ and $t1_f = t2_f$): This dependency establishes that two tasks must occur simultaneously. It is the same relation originally proposed by Allen.

T1 starts T2: This relation has been divided into two.

T1 startsA T2 ($t1_i = t2_i$ and $t1_f < t2_f$): Both tasks must start together and the first must finish first. It is the original relation.

T1 startsB T2 ($t1_i = t2_i$): Variation of the original relation, relaxing the obligation that the first task must finish first. This variation makes sense because in some situations it is required that both tasks start together, but it does not matter when they will finish.

T1 finishes T2: Similarly to the previous one, it is possible to define two relations based on it.

T1 finishesA T2 ($t1_i > t2_i$ and $t1_f = t2_f$): Both tasks finish together, but the first must start after the second. It is the original relation.

T1 finishesB T2 ($t1_f = t2_f$): Similarly to startsB, this dependency is obtained from the original, relaxing the restriction that T1 must start after T2. This dependency is important for situations in which it does not matter when tasks have begun, but they must finish simultaneously.

T1 before T2: This relation clearly illustrates the difference between Allen's temporal logic and task interdependencies. It can be divided into three distinct interdependencies.

T2 afterA T1 ($t1_{f,n} < t2_{i,n}$, $\forall n > 0$, where *n* means the n^{th} execution of the task): T2 may only be executed if T1 has already finished (the restriction occurs in the execution of T2; T1 can be freely executed). This dependency is the prerequisite relation, which is very common in collaborative applications. In this case, T2 may be executed only once after each execution of T1.

T2 afterB T1 ($t1_{f,1} < t2_{i,n}$, $\forall n > 0$): Variation of the previous dependency, in which T2 may be executed several times after a single execution of T1.

T1 beforeA T2 ($t1_f < t2_i$): From a temporal logic point of view, this relation is the opposite of *after* (the formal definition is the same). However, they generate totally different coordination mechanisms. Essentially, the difference is that in this case the restriction occurs in the execution of T1, which may not be executed anymore if T2 has already started its execution. There is no restriction on the execution of T2 (T2 does not have to wait for the execution of T1, as would happen with the dependency *T2 afterA T1*).

T1 meets T2 ($t1_f = t2_i$): T2 must start immediately after the end of T1. It is the original relation.

T1 overlaps T2: This relation is divided into two types.

T1 overlapsA T2 ($t1_i < t2_i < t1_f < t2_f$): T1 starts before T2 and T2 must start before the end of T1, which must finish before T2. It is the original relation.

T1 overlapsB T2 ($t1_i < t2_i < t1_f$): Variation of the original relation, in which it does not matter which task finishes first. The only obligations are that T1 starts before T2 and T2 starts before the end of T1.

T1 during T2: This relation is also adapted to generate two new interdependencies.

T1 duringA T2 ($t1_{i,n} > t2_{i,n}$ and $t1_{f,n} < t2_{f,n}$, $\forall n > 0$): T1 must be totally executed during the execution of T2. In this case, a single execution of T1 is allowed during an execution of T2.

T1 duringB T2 ($t1_{i,n} > t2_{i,m}$ and $t1_{f,n} < t2_{f,m}$, $\forall m > 0$ and $\forall(n \geq m)$): Variation of the previous dependency, in which T1 may be executed more than once during a single execution of T2.

A consequence of the included redundancies is that there is not a unique way to represent interdependencies among tasks, but these redundancies give a more manageable and understandable perspective to the model.

3.1.2. Resource management interdependencies. It has been shown that combinations of Allen's basic relations could also represent resource-related interdependencies [24]. For example, if two tasks, T1 and T2, may not use the same resource simultaneously, it is possible to define a "not parallel" dependency as the following statement: *T1 afterA T2* or *T1 meets T2* or *T2 afterA T1* or *T2 meets T1*. However, it is important to reinforce the point that the context of interest is that of collaborative activities, in which the notion of resource is very strong. Therefore, it is not reasonable to ignore this notion and 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 work with each kind of dependency separately.

Resource management interdependencies in the presented framework 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. An example is when two or more users simultaneously want to alter the same part of a document in a collaborative authoring system [18]. Three basic resource management dependencies are defined.

Sharing: A limited number of resources must be shared among several tasks. It represents a common situation that occurs, for example, when several users want to edit a document. This dependency includes the notions of *divisibility* and *concurrency* of the genres coordination proposal previously discussed [25].

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.

From the basic interdependencies discussed above, it is possible to define composite interdependencies.

Sharing M + simultaneity N: Represents the situation in which up to *M* groups of *N* tasks may share a resource.

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

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

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

Different than 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 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).

3.2. Dimensions of interdependencies

There is no claim here that the set of interdependency types presented above is complete. Such affirmation could only be made if it was possible to express with this model all existent collaborative scenarios for all possible contexts. Once it is impossible to enumerate all scenarios, it must be learned from experience, using the model in several collaborative scenarios in order to extend the

framework. One of the improvements brought about by the use of the framework is the inclusion of what has been called "dimensions" of interdependencies.

During the creation of coordination mechanisms associated with the described interdependencies, the necessity to treat some specific situations that were common to all interdependencies was perceived. These specific situations have been organized in the framework as the referred dimensions, which are additional characteristics of the basic interdependencies that constitute guidelines for the identification of adequate coordination mechanisms. Currently, two dimensions have been identified, *activeness*/*passiveness* and *imprecision*.

3.2.1. Activeness/passiveness. The same set of interdependencies may generate completely different sets of coordination mechanisms if they are considered active or passive interdependencies. This is especially true for temporal interdependencies and simultaneity (resource management dependency).

In the active situation, a task "forces" the execution of its interdependent tasks in order to achieve the desired behavior. Here, the notion of prerequisite (i.e., a task being ready) is not so strong as the order given by interdependent tasks. In other words, it is the execution of a task that determines the execution of others. An example is when a user makes a change in a WYSIWIS application, which must be immediately propagated to all other users sharing this application. In this case, tasks are related by the *startsB* interdependency, indicating that the execution of one enacts the execution of others. A similar situation could happen for simultaneity interdependency. If a task wants to use a resource, it may start the execution of other(s) that also use(s) the same resource in order to reach the number of tasks needed to make use of the resource.

It is possible to say that, for the active situation, coordination mechanisms are built for guaranteeing that tasks will be executed guided by their interdependencies. Conversely, for the passive situation, coordination mechanisms are built for guaranteeing that interdependencies will not be violated, but they do not guarantee that all tasks will be executed.

In the passive situation, a task does not force the execution of interdependent tasks. When a task becomes ready to execute but there is an interdependent task that does not enable the validation of the interdependency, it simply waits until the other also gets ready. For example, in the *startsB* interdependency, if the first task is ready, but not the other, they will only be executed when the second gets ready. Here, the notion of prerequisite in relation to the execution of tasks prevails.

The use of passive coordination mechanisms has shown that there could be a number of situations subject to deadlocks, especially if the task belongs to complex activities. For example, in relation *T1 equals T2*, the agent responsible for the execution of T2 (i.e., who realizes the task – a human user, a software agent, etc.) could have an alternative path that does not execute T2. In this case, the agent responsible for the execution of T1 could be blocked, waiting indefinitely for T2. To minimize this kind of problem the use of timeouts in the coordination mechanisms is proposed [19]. Two kinds of timeouts are defined, which may be used in coordination mechanisms for all interdependencies previously presented. In the first kind (called *timeoutA*), an alternative task is executed after a certain waiting period if the original task has not yet been executed. This kind of timeout can be thought as an "emergency procedure" to avoid that other tasks be blocked. The second kind of timeout (*timeoutB*) sends tasks back to their initial states after a certain waiting period instead of executing an alternative task. The return to the task initial state makes sense because, before starting its execution, a task could have a resource assigned to it. Therefore, it is necessary to release this resource. TimeoutB is less "aggressive" than timeoutA, in the sense that it does not overpass the interdependency, but it only works when the agent responsible for the execution of the blocked task has an alternative path that does not execute that task.

Another possible consequence of passive coordination mechanisms is that the non-execution of an expected task may invalidate previous tasks. An example occurs in relation *T2 afterA T1*. T1 may be executed without restrictions, but in some cases it expects the execution of T2 to be "validated." A passive coordination mechanism, however, does not guarantee that T2 will ever be executed. For this reason, after a certain waiting period the coordination mechanism should be able to warn the agent responsible for executing T1 that its execution was invalidated. This situation occurs, for example, in e-commerce, where the processing of an order must be followed by the payment. If the payment is not realized, after a certain period the order should be canceled.

3.2.2. Imprecision. The use of the presented coordination model has shown that it is sometimes very difficult to completely define the interdependencies underlying collaborative activities. This happens because these relationships already may embed in their essence a not so well defined (or fuzzy) semantics. This occurs, for instance, when a second task wants to start its execution when a first task is "almost finishing." This scenario is not accepted by conventional temporal interdependencies, since tasks may only be synchronized by their starting or finishing times. Such modeling imprecision is important because it offers application designers a degree of flexibility through which they may 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. In particular, a fuzzy Petri net-based approach has been used for this purpose [20].

Taking into account temporal interdependencies, the use of fuzzy coordination mechanisms may enhance the degree of parallelism in the execution of tasks. At one side, this happens because a task that would have to wait for another to get ready may now start its execution when the other is not completely ready. At the other side, it is possible to define synchronization points before the actual end of the tasks (i.e., if a first task is a prerequisite for a second one, the latter can be able to start before the actual end of the former). To realize these situations, consider a hypothetical toy manufacturing system. In order to assemble a doll, its legs and arms must be produced at the same time by different machines. Therefore, it may be said that both tasks have the *equals* interdependency. However, if the production of legs takes a little longer, this task could be started a bit before the production of arms. Consider now a third task, the assembly of the doll, which requires the arms and legs resulting from the previous tasks. In the conventional situation, this task may start only after the end of the first two tasks. However, if there is an initial phase of this task that does not require the legs and arms (e.g., the setup of the assembly machine) it may be started when those pieces are, say, 80% ready.

Another aspect of fuzzy coordination mechanisms is that they significantly reduce the number of deadlock situations, because a task not completely ready may enable the execution of another.

For all these reasons, the use of fuzzy coordination mechanisms may bring a higher degree of flexibility and manageability to CSCW that should be explored.

3.3. Global vision

Figure 2 summarizes the framework presented in this section. On a more abstract level, the interdependencies among tasks are defined, separated into temporal and resource management dependencies. On the middle level, additional characteristics (dimensions) of the interdependencies are identified for each situation. Then, based on the interdependencies and their dimensions, it is possible to build (or reuse) adequate coordination mechanisms.

A set of coordination mechanisms based on this framework, that uses Petri Nets (PNs) [16] as a modeling tool has been presented elsewhere [17], [19]. The next section shows a case study that uses some of these coordination mechanisms for the implementation of a collaborative videogame.

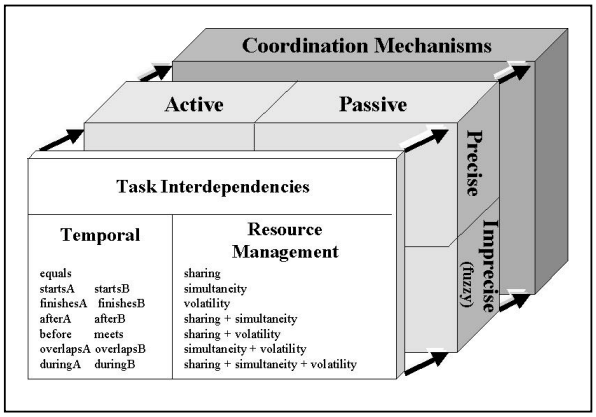

Figure 2. Illustrative representation of the proposed framework.

4. Case study

In Collaborative Virtual Environments (CVEs), users are simultaneously present and may interact with objects and other users. Currently, the development of CVEs has been dominated by leisure activities, basically enabling navigation through virtual scenarios and communication with remote users [8]. This kind of activity is well coordinated by the "social protocol," that is characterized by the absence of any coordination mechanism, trusting the participants' abilities to mediate interactions. Therefore, there are a growing number of studies focusing on interactions in CVEs (e.g., [15]). However, activities related to cooperative work also require sophisticated coordination mechanisms to be efficiently realized in this kind of system. For this reason, CVEs are potential targets for the proposed coordination model.

In this section a case study of a CVE where a user interacts with an autonomous agent that represents a second user is presented. The example implements a kind of videogame based on the second "task" (activity) of Heracles, from the Greek mythology. According to the legend, Heracles had to kill the Hydra of Lerna, a monster with nine heads that are regenerated after being severed. In order to achieve his goal, Heracles needs the collaboration of his nephew Iolaus, who cauterizes the monster's wounds after Heracles cuts off each head. However, the last head cannot be severed by any weapon. The solution is to bury the monster in a deep hole and cover it with a huge stone.

Figure 3 illustrates an abstract PN-like model of the videogame (open rectangles indicate interdependent tasks). There are two identical nets, one representing the user's and the other representing the agent's sequence of tasks. Each net has two alternative paths, indicating that

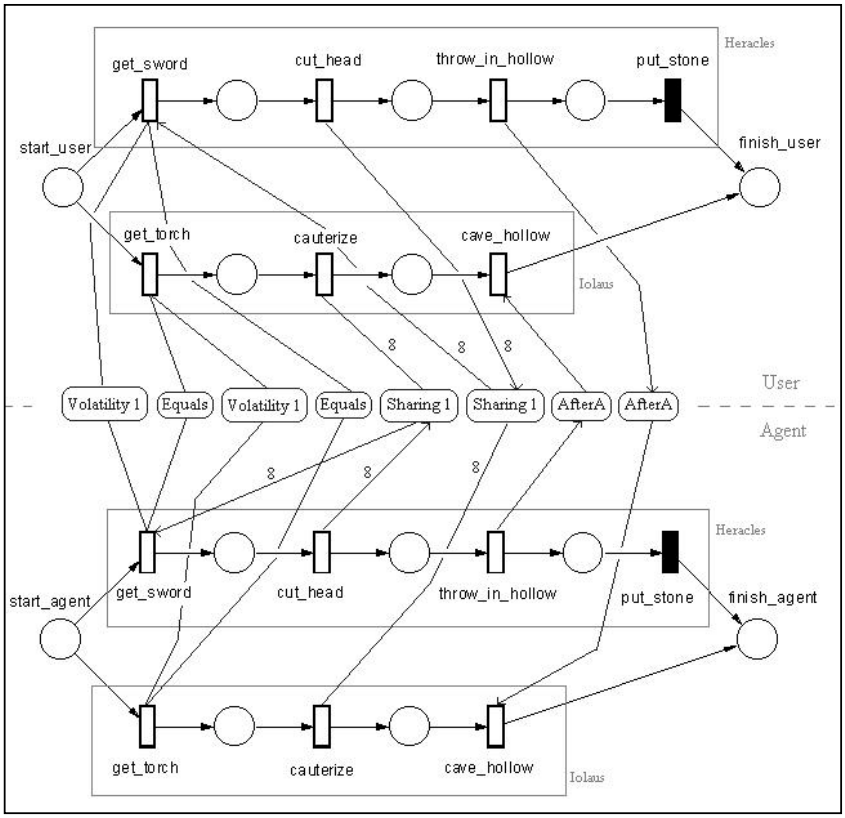

Figure 3. Model of Heracles videogame.

each "actor" (user or agent) may assume either role (Heracles or Iolaus). The upper part of the nets represents Heracles' sequence of tasks. He must get the sword, sever eight of the Hydra's heads, throw the beast into the hole and cover it with a stone. The lower part of the nets represents Iolaus' sequence of tasks. He must get the torch, cauterize the wounds after Heracles has severed the heads and dig the hole.

The definition of which actor will assume which role is given by the interdependencies *volatility 1* between the tasks get_sword and get_torch. Since there are only one sword and one torch available, the weapon's choice determines that each actor will assume a different role. There is also an *equals* interdependency between get_sword and get_torch of different actors. This interdependency requires an active coordination mechanism in order to force the agent to choose the other weapon when the user chooses his/her weapon.

The interdependency *sharing 1* among the tasks get_sword, cut_head and cauterize is the "core" of the game. When Heracles gets the sword, he is also assigned eight resources that may be thought of as "abstract authorizations" to cut Hydra's heads. After he has severed each head, a resource is released, indicating to Iolaus that he may cauterize that wound. This dependency requires a passive coordination mechanism, because it is not necessary to force Iolaus to cauterize each head after it has been severed. However, if the head wound is not cauterized within a certain period after it has occurred, this task returns to its initial state (timeoutB) and also reassigns the resource to Heracles, indicating that he must once again sever that head (the head is regenerated).

There is also an interdependency *afterA*, indicating that Heracles may only throw the monster in the hole if Iolaus already has dug it. For this dependency, a passive coordination mechanism was used.

The coordination mechanisms for the above interdependencies were implemented using PNs. This choice was made because, besides being easy to understand, PNs offer a strong theoretical support for the analysis of an environment's behavior and supplementary simulation techniques. Using the PN-based model, it is possible to anticipate and test the behavior of collaborative environments before their implementation. A detailed description of the implemented coordination mechanisms and the modeling and analysis of the game

has been presented elsewhere [19]. This example used only precise coordination mechanisms, modeled by "conventional" or high-level PNs, but imprecise ones could be used by means of fuzzy PNs.

The videogame was implemented using the blaxxun Contact [4], a client for multimedia communication that provides resources for VRML (Virtual Reality Modeling Language) visualization. The interaction with the user occurs by means of buttons defined in a Java applet that interacts with the VRML world via EAI (External Authoring Interface), an interface that enables external programs to interact with objects of a VRML scene. By clicking on the applet's buttons, the user orders the execution of a task in the virtual world. Therefore, the coordination mechanisms act on the interface's buttons, enabling or disabling them if their respective tasks are enabled or not. In order to make the game more dynamic, the agent's behavior is aleatory, taking a variable amount of time to begin the execution of the tasks imputed to it. For example, when the user assumes the role of Heracles, the agent (Iolaus) may not cauterize a head severed by Heracles before it is regenerated. Figure 4 shows some frames of the videogame (the nine heads of Hydra are represented by nine monsters). Frames *a* and *b* show Heracles' interface, while frame *c* shows Iolaus' interface.

The implementation of the videogame demonstrated that, in order to be potentially reusable, coordination mechanisms require a standard interface with the system responsible for the execution of interdependent tasks. The definition of such an interface is one of the next steps of this work, as discussed in the following section.

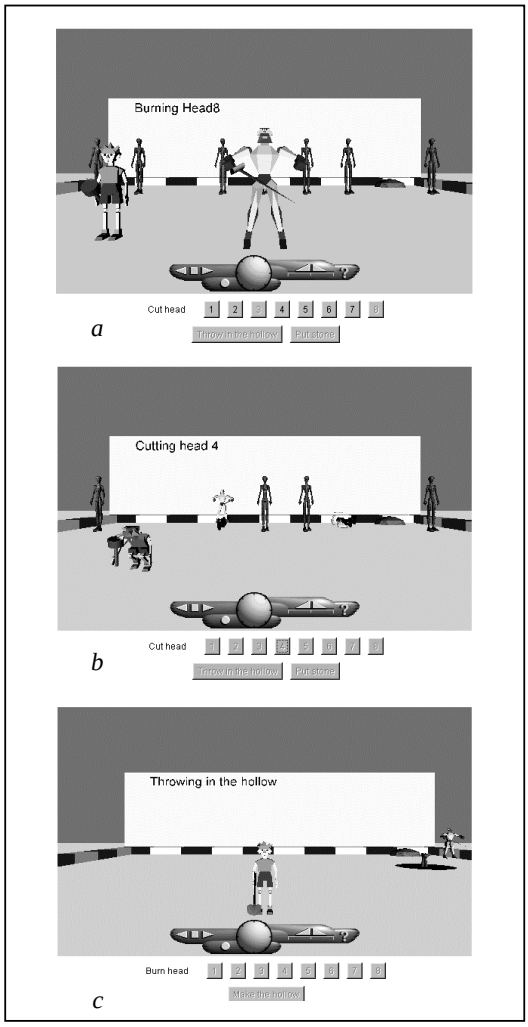

Figure 4. Frames of the Heracles videogame.

5. Conclusions

This paper introduced a framework for the definition of interdependencies among tasks in collaborative activities. The framework also defines additional characteristics for the interdependencies (the dimensions) that show that it is possible to have several coordination mechanisms for the same basic task interdependency. The interdependencies and their dimensions provide guidelines for the identification of the most appropriate coordination mechanism in each situation.

The coordination model suggested by the presented framework fits within the second generation because it offers a degree of flexibility through separation of tasks and interdependencies and is adequate for dealing with interoperability aspects. This is so in the sense that the interdependencies are generic (i.e., may be applied in 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 in the example and in the cited references, the framework clearly separates interdependencies from their coordination mechanisms, enabling the use of different implementation tools for the coordination mechanisms.

The set of interdependencies presented in the framework does not claim to be complete, since it would be very difficult to establish a framework with all possible interdependencies between tasks. However, the framework is extensible, in the sense that new interdependencies and dimensions may be added to it, enlarging the set of situations it encompasses.

A next step of this work is the implementation of software components to implement the coordination mechanisms. The component model will standardize an event-based interaction between tasks and associated coordination mechanisms in an implementation independent manner. The implemented PN-based

mechanisms [19] may be one of the possible kernels of these components.

The proposed framework will also constitute the basis of an educational tool that plans to use CVEs as experimental learning mechanisms.

Finally, it is important to reinforce that the coordination of interdependent tasks in collaborative activities is a problem that should be addressed to ensure the effectiveness of the collaboration. The separation between tasks and interdependencies, and a further association between interdependencies and adequate coordination mechanisms are important goals in this direction. The framework presented here is a contribution towards these goals.

Acknowledgments. A. Raposo is sponsored by FAPESP (Foundation for Research Support of the State of São Paulo, Brazil), grant n. 00/10247-3. H. Fuks has a researcher productivity allowance from CNPq (Brazilian National Research Council), grant n. 524557/96-9. Thanks also to A. Coelho and A. Cruz for their helpful insights.

6. References

[1] J. F. Allen, "Maintaining Knowledge about Temporal Intervals", *Comm. of the ACM*, 26(11): 832-843, Nov 1983.

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

[3] P. C. Attie et al., "Scheduling workflows by enforcing intertask dependencies", *Distributed Systems Engineering Journal*, 3(4): 222-238, Dec 1996.

[4] blaxxun interactive, *blaxxun Contact 4.4*, <http://www.blaxxun.com/ products/contact>, Aug 2000.

[5] M. Cortes, "A Coordination Language For Building Collaborative Applications", *Computer Supported Cooperative Work*, 9(1): 5-31, 2000.

[6] W. K. Edwards, "Policies and Roles in Collaborative Applications", *Conf. on Computer Supported Cooperative Work (CSCW´96)*, pp. 11-20, 1996.

[7] C. A. Ellis, and J. Wainer, "A Conceptual Model of Groupware", *Conf. on Computer Supported Cooperative Work (CSCW´94)*, pp. 79-88, 1994.

[8] E. Frécon, and A. A. Nöu, "Building Distributed Virtual Environments to Support Collaborative Work", *Symp. on Virtual Reality Software and Technology (VRST´98)*, pp. 105-119, 1998.

[9] D. Gelernter, and N. Carriero, "Coordination Languages and their Significance", *Comm. of the ACM*, 35(2): 97-107, Feb 1992.

[10] W. Hasselbring, "Information System Integration", *Comm. of the ACM*, 43(6): 33-38, Jun 2000.

[11] T. W. Malone, and K. Crowston, "What is Coordination Theory and How Can It Help Design Cooperative Work Systems?", *Conf. on Computer-Supported Cooperative Work (CSCW'90)*, pp. 357-370, 1990.

[12] T. W. Malone, and K. Crowston, "The Interdisciplinary Study of Coordination", *ACM Computing Surveys*, 26(1): 87-119, Mar 1994.

[13] T. W. Malone, K.-W. Lai, and C. Fry, "Experiments with Oval: A Radically Tailorable Tool for Cooperative Work", *ACM Trans. Information Systems*, 13(2): 177-205, Apr 1995.

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

[15] T. Manninen, "Interaction in Networked Virtual Environments as Communicative Action: Social Theory and Multi-player Games", *6th Int. Workshop on Groupware (CRIWG'2000)*, pp. 154-157, 2000.

[16] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proc. of the IEEE*, 77(4): 541-580, Apr 1989.

[17] A. B. Raposo, L. P. Magalhães, and I. L. M. Ricarte, "Petri Nets Based Coordination Mechanisms for Multi-Workflow Environments", *Int. J. Computer Systems Science & Engineering*, 15(5): 315-326. Sp. Issue on Flexible Workflow Technology Driving the Networked Economy, Sep 2000.

[18] A. B. Raposo, L. P. Magalhães, and I. L. M. Ricarte, "Coordination Mechanisms for Collaborative Environments", *VI Simpósio Brasileiro de Sistemas Multimídia e Hipermídia (SBMIDIA'00)*, pp. 247-258, 2000. In Portuguese.

[19] A. B. Raposo, *Coordination in Collaborative Environments Using Petri Nets*, Ph.D. Thesis, DCA – FEEC – UNICAMP, October 2000. In Portuguese.

[20] A. B. Raposo, A. L. V. Coelho, L. P. Magalhães, and I. L. M. Ricarte, "Using Fuzzy Petri Nets to Coordinate Collaborative Activities", Accepted to *Joint 9th IFSA (International Fuzzy Systems Association) World Congress and 20th NAFIPS (North American Fuzzy Information Processing Society) Int. Conf.*, 2001.

[21] K. Schmidt, and L. J. Bannon, "Taking CSCW Seriously – Supporting Articulation Work", *Computer Supported Cooperative Work*, 1(1-2): 7-40, 1992.

[22] K. Schmidt, and C. Simone, "Coordination mechanisms: Towards a conceptual foundation of CSCW systems design", *Computer Supported Cooperative Work*, 5(2-3): 155-200, 1996.

[23] C. Simone, G. Mark, and D. Giubbilei, "Interoperability as a Means of Articulation Work", *Proc. Int. Conf. on Work Activities Coordination and Collaboration (WACC'99)*, pp. 39-48, 1999.

[24] T. Wahl, and K. Rothermel, "Representing Time in Multimedia Systems", *Proc. IEEE Conf. on Multimedia Computing and Systems*, pp. 538-543, 1994.

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