PERGAMON

Computers & Graphics 25 (2001) 1025–1039

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

# Coordination components for collaborative virtual environments

Alberto B. Raposo[a,*], Adailton J.A. da Cruz[a,b], Christian M. Adriano[a], Léo P. Magalhães[a]

[a] *Department of Computer Engineering and Industrial Automation (DCA), School of Electrical and Computer Engineering (FEEC), State University of Campinas (UNICAMP), CP 6101, 13083-970, Campinas, SP, Brazil*

[b] *University Center of Dourados, Federal University of Mato Grosso do Sul, CP 332, 79804-970, Dourados, MS, Brazil*

**Abstract**

This paper deals with the behavior of virtual environments from the collaboration point-of-view, in which actors (human or virtual beings) interact and collaborate by means of interdependent tasks. In this sense, actors may realize tasks that are dependent on tasks performed by other actors, while the interdependencies between tasks (through resource management and temporal relations) delineate the overall behavior of a virtual environment. Our main goal is to propose an approach for the coordination of those behaviors. Initially a generic study of possible interdependencies between collaborative tasks is presented, followed by the formal modeling (using Petri Nets) of coordination mechanisms for those dependencies. In order to implement such mechanisms, an architecture of reusable and pluggable coordination components is also introduced. These components are used in an implementation of a multi-user videogame. The presented approach is a concrete step to create virtual societies of actors that collaborate to reach common goals without the risk of getting involved in conflicting or repetitive tasks. © 2001 Elsevier Science Ltd. All rights reserved.

*Keywords:* Modeling of behavior; Collaborative virtual environment; Coordination; Computer-supported cooperative work; Software components

## 1. Introduction

We are currently witnessing the rapid establishment of virtual societies, in which remote interaction is a feasible alternative to face-to-face contact, transcending geographic location and time constraints (“anywhere, anytime”) [1]. Two of the technological driving forces in the virtual societies are computer-supported cooperative work (CSCW) and networked virtual environments (net-VEs).

CSCW is defined as “an endeavor to understand the nature and characteristics of cooperative work with the objective of designing adequate computer-based technologies [to support this kind of activity]” [2]. In other words, CSCW is interested in creating systems to support groups of people engaged in tasks with a common goal (i.e., collaboration).

A net-VE is a simulation of a real or imaginary world where multiple users may interact with one another in “real-time”, share information, and manipulate objects in the shared environment [3,4]. Net-VEs surpass the desktop metaphor of most current applications, proposing virtual communities where interactions are modeled according to the interactions in the real world.

Owing to their great potential as tools for CSCW, net-VEs have been developed with an eye on CSCW results.

*Corresponding author. Tel.: +55-19-3788-3720; fax: +55-19-3289-1395.

*E-mail addresses:* alberto@dca.fee.unicamp.br (A.B. Raposo), ajcruz@dca.fee.unicamp.br (A.J.A. da Cruz), medeiros@dca.fee.unicamp.br (C.M. Adriano), leopini@dca.fee.unicamp.br (L.P. Magalhães).

This is particularly true for aspects such as user awareness, user embodiment and spatial models of interaction, for which results from the CSCW field have been successfully implemented in net-VEs [5–7]. When the net-VEs are aimed at collaborative activities, they are also called collaborative virtual environments (CVEs).

In spite of CVEs' suitable characteristics, there is still a gap between these environments and CSCW regarding the coordination of their activities. 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 what we call "loosely coupled collaborative activity" and is well coordinated by a social protocol, characterized by the absence of any explicit coordination mechanism, trusting the participants' abilities to mediate interactions.

On the other hand, "tightly coupled collaborative activities" require sophisticated coordination mechanisms in order to be efficiently performed in CVEs. In this kind of activity, tasks depend on one another to start, to be performed, and/or to end. A very illustrative example comes from the field of flight control and its tightly coupled activities of coordinating air traffic. Several technologies are under test to replace flight controllers' protocols for fully computer-based coordination solutions [9]. Other examples of tightly coupled activities may be found in collaborative authoring, workflow procedures, and multi-user computer games. Interdependencies among tasks in this kind of activity are normally positive, in the sense that each actor wants the others to succeed. However, they are not always harmonious. There must be coordination between tasks in order to ensure collaboration effectiveness. Without coordination, there is a risk that actors may 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" [10] and is enacted by coordination mechanisms [11] that are software devices that interact with the application to control the behavior of a virtual environment.

The work presented in this paper is a step towards shortening the gap between CVEs and CSCW, by the introduction of an architecture of coordination components that may be used to control tightly coupled collaborative activities performed by means of CVEs. These components coordinate a set of interdependencies that often takes place between collaborative tasks, ensuring that these dependencies will not be violated. By using pluggable coordination components, different behaviors may be applied to the same CVE, just by changing components. Moreover, the components are generic and may be reused in other CVEs.

The following section overviews some related work. In Section 3, our coordination approach is presented. The approach is divided into three parts. Initially a generic set of interdependencies is defined. Then, the formal modeling of the coordination mechanisms is presented, using Petri Net as modeling and simulation tool. Finally, the coordination components architecture is depicted and its integration with the virtual environment is thoroughly discussed. Section 4 exemplifies the approach by means of implementing a multi-user videogame. Conclusions and suggestions for future research are drawn in Section 5.

## 2. Related work

This paper comprises three distinct areas of study, namely, CVEs, coordination (from the CSCW point-of-view) and software components. For this reason, we initially present a separate overview of related aspects of each of those areas and then explain how they are joined together in our approach.

### *2.1. Virtual environments*

Net-VEs are not a recent technology. Experimental systems have been around for decades, but only recently did they start to gain ground outside academic and military units. This popularity increase is mainly due to the rapid development of computing power and networking technologies, as well as their costs reduction.

In parallel to their growing popularity, a number of challenges remain to be overcome. Amongst them, we could mention all the problems related to management of network resources (concurrency, data loss, network failure, scalability, etc.); all the problems related to real-time graphics applications (e.g., CPU allocation for rendering); and those related to interactive multi-user applications (real-time data I/O, consistency among users, etc.) [4]. Additionally, there are also specific problems related to the application field of the net-VE, such as integration with large databases (e.g., geographic information about a terrain for military training environments); persistent storage (e.g., for engineering applications); and user authentication (e.g., for commerce applications).

When the application field is specifically collaboration, all the challenges related to CSCW should be also added to the challenges above. Just to name a few of them, there is the difficulty in relating spatial considerations to social interaction [5]; the difficulty in assisting individuals in working flexibly with virtual workplace objects [12]; and the necessity to create realistic avatars to improve communication among participants and their sense of presence [13].

The design of a CVE encompasses three distinct but interrelated aspects, namely, *form*, *function* and *behavior* [14]. Form is related to objects appearance, structure and physical properties, as well as the scene structure of the virtual world. Function refers to what objects do to accomplish their behavior (i.e., the actions they execute in the virtual world). Behavior refers to how objects define and dynamically change the different functions that they carry out over a period of time.

In this paper, we have been specifically concerned with behavioral aspects of CVEs. Although function and behavior are deeply related, the coordination approach to be presented clearly separates tasks (function) from task interdependencies and coordination components (behavior). One of the advantages of this approach is the possibility of altering behaviors by simply altering the coordination components without having to alter the core of the CVE. For example, when a certain event takes place, the behavior of an actor might define in which direction it should walk. This behavior may be altered to define the direction in which the actor should run from then on, when the same event occur again. The executions of the tasks run and walk are independent from the coordination system, being related only to the form (e.g., the walk of a biped actor is different from that of a quadruped one).

## 2.2. Coordination in CSCW

Only in the second half of the 80s have collaborative systems with some kinds of coordination mechanisms started to appear. However, they were restricted to specific scenarios, because coordination protocols were rigidly defined, restraining dynamic modifications. 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 on the coordination model.

The idea of creating a set of tasks interdependencies and respective coordination mechanisms was proposed in the coordination theory of Malone and Crowston [10]. They defined three types of elementary resource-based dependencies and worked with the hypothesis that all other dependencies could be defined as combinations or specializations of these basic types. One of the advantages of this approach is the possibility to alter coordination policies simply by altering the coordination mechanisms for the interdependencies. 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, by creating a set of interdependencies and respective coordination mechanisms capable of encompassing a wide range of collaborative applications.

Another use of interdependencies lies in the management of workflow activities [15]. 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 ensuring 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 paper starts with some of the ideas of these previous works, and it is refined by defining a larger set of basic interdependencies that includes both temporal and resource management dependencies.

## 2.3. Software components

The first attempt to conceptualize the software components paradigm was directed by the concern with reuse and modularization [16]. Components were idealized based on an analogy with electronic systems, which are characterized by reusable modules with clear functions. This paradigm was forgotten for many years and then reappeared in a new software development context.

Nowadays, issues such as autonomy, interconnection and integration complement the component approach, in addition to reuse and modularization [17]. The paradigm advocates the idea that a component-based system should be composed of independent and autonomous parts compromised with their mutual decoupling (i.e., they are integrated in order to provide complete functions, but they do not either keep references to one another or communicate directly—the communication is achieved by means of messages). The decoupling compromise satisfies a number of requirements, such as the substitution of a component for another and the immediate insertion of new components (extensibility).

Regarding its fundamental elements, the component paradigm is defined by means of a software architecture [18], in which graph nodes are called components and the arcs are called connectors. Components are said to be active because they are responsible for all processing. Each component has a complete and self-contained service that is delivered through a specific interface pattern. By means of this pattern the component externalizes events and messages while still keeping its encapsulation. Connectors forward events and messages, and establish a loosely coupled integration between software components.

We have been implementing the component-based coordination architecture as a set of JavaBeans [19], which is a framework consisting of a Java API that enables the implementation of software components by the rules stated for the architectural style mentioned above. Software components are named beans, and connectors are objects of classes that extend appropriate event listeners.

For all the reasons discussed above, the component model is a very satisfactory solution for the design of CVEs, especially those supporting a task/interdependency model for tightly coupled collaborative activities.

As to the components' design, our choice for JavaBeans instead of other frameworks (e.g., DCOM) is motivated by the facility for integrating the Java language with Web-based CVEs.

### *2.4. Components as coordination mechanisms for CVEs*

Our main goal is to create facilities for the design and implementation of CVEs. As already mentioned, coordination mechanisms are important in the use of CVEs for executing tightly coupled collaborative activities. The implementation of these mechanisms as components allows for the creation of an application-independent library of coordination components that permits incremental modifications to virtual environments. The application of component-based architectures in CSCW is a recent approach. One of the few works on this topic [20] extends the JavaBeans architecture to create a framework for building collaborative infrastructures.

Another important aspect of our approach is the development of a formal modeling of collaborative environments [21]. We used a Petri Net (PN) based modeling that enables the designer to anticipate and test the behavior of CVEs before their implementation, avoiding the trail and error approach. 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 the main features of CVEs, 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. There are work results that also use PNs for modeling, analysis and simulation of net-VEs [22,23], however, they are more related to physical aspects of CVEs implementation, such as tracking user's position and displaying the walls in an immersive environment. To our knowledge, there is no other work that uses PNs for simulation and analysis of actor's behaviors in CVEs (this idea has originated from a previous approach that uses PNs to model computer animations [24]).

## 3. The coordination approach

Before presenting our coordination model, it is necessary to clarify the definition of task used in this paper. In the present context, tasks are the "building blocks" of a collaborative activity, which is defined as a coordinated set of tasks performed by multiple actors to achieve a common goal. Tasks may be atomic or composed of subtasks and are connected to one another through interdependencies, and the management of such interdependencies is effected by coordination mechanisms. The granularity of a task reflects the interdependencies the current task 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) may be nestled as a single task. For example, in a lower abstraction level, the walk of a bipedal structure may be considered a collaborative activity. In this activity, each limb may be considered an actor whose tasks are the desired movements that have interdependencies with the movements of other limbs (e.g., both legs may not be out of the ground at the same time). On the other hand, in a higher abstraction level, the walk may be considered a single task of a bipedal actor engaged in a more complex collaborative activity (e.g., a videogame).

Using this flexible definition of task, it is possible to model collaborative activities in several abstraction levels, which improves both the understandability and the feasibility of the interacting rules that characterizes the whole process.

The present section has three parts, namely, the definition of a set of frequent interdependencies between cooperative tasks; the formal modeling of coordination mechanisms to control these interdependencies; and the development of a library of coordination components to implement the modeled mechanisms.

### *3.1. Interdependencies*

Interdependencies are divided into two types, *temporal* and *resource management*. This separation agrees with the coordination model proposed by Ellis and Wainer [25]. According to their model, the coordination in collaborative systems could occur in two levels, activity level and 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 interdependencies establish the execution order for tasks. The set of temporal interdependencies of our coordination model is based on temporal relations defined by Allen [26]. 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 to task coordination purposes, because tasks are generally non-instantaneous operations.

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 of making 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 a temporal logic's point-of-view, but makes the coordination model more manageable. The result of the adaptation of Allen's relations to the context of collaborative activities is the set of 13 temporal interdependencies presented below [27].

Considering 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.

*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 end before. This is the original relation proposed by Allen.

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

*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 end together, but the first must start after the second. This 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, as long as they end 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 after 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 ended (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 to *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 further executed if T2 has already started its execution. There is no restriction to the execution of T2 (T2 does not have to wait for the execution of T1, as it would happen to the dependency *T2 afterA T1*).

*T1 meets T2* ($t1_f = t2_i$): T2 must start immediately after the end of T1.

*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 end 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 ends 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 $\forall (n \geqslant m)$): 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 in Allen's logic is that there is not a unique way to represent interdependencies among tasks, but these redundancies give a more understandable and manageable perspective to the model.

### *3.1.2. Resource management interdependencies*

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

*Sharing*: A limited number of resources may be shared among several tasks. It represents a common situation that occurs, for example, when several users want to edit a document.
*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 its use, the resource is available again. For example, a printer is a non-volatile resource, which a sheet of paper is volatile.

From the basic interdependencies above, it is also possible to define composite interdependencies. For example, sharing $M$ + volatility $N$ indicates that up to $M$ tasks may share a resource, which may be used $N$ times only.

Different from temporal dependencies, resource management dependencies are not binary relations. It is possible, for example, that more than two tasks share a resource. Moreover, each of the above 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. Modeling coordination mechanisms*

The mechanisms to coordinate the above set of interdependencies were modeled by using PNs. The formal modeling of the mechanisms not only does enable a previous verification and validation of the CVE's behavior, but also constitutes the internal logic of the coordination components (which will be presented in the next section).

In the proposed scheme, the design of a collaborative environment is divided into three distinct hierarchical levels, *workflow*, *coordination* and *execution* (Fig. 1).

At the *workflow* level, a global sketch of the environment's behavior is delineated, establishing which elements of the scene will be considered actors and which tasks are assigned to them. Each actor's behavior is modeled separately, establishing the interdependencies between tasks of the same actor or those of different actors.

The *coordination* level is built under the workflow level by the expansion of interdependent tasks according to a PN-based model defined in [28] and the insertion of correspondent coordination mechanisms between them. At this level, we have a complete PN model of the actors' behaviors, allowing anticipation of undesired situations (such as deadlocks) through simulation, verification, validation and performance analysis of the model.

The *execution* level deals with the actual execution of tasks in the CVE. This level is the interface between the coordination infrastructure and the CVE. It will be treated in the next section.

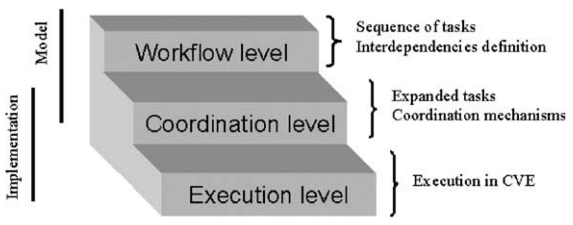

Fig. 1. Hierarchical levels for the design of CVEs.

During the passage from the workflow to the coordination level, each task that has a dependency on another is modeled by a system with five transitions (*ta*, *tb*, *ti*, *tf*, and *tc*) and four places (*P*1, *P*2, *P*3 and *P*4), as proposed by van der Aalst et al. [28]. As shown in Fig. 2, attached to each expanded task, there are also five places that represent the interaction with the resource manager, the temporal coordination mechanism, and the agent that executes the task. The places *request_resource*, *assigned_resource* and *release_resource* connect the task with the resource manager. The places *start_task* and *finish_task* connect the task with the temporal coordination mechanism and the agent that performs it, respectively, indicating the beginning and the end of the task execution.

In order to illustrate a temporal coordination mechanism, Fig. 3. presents the model of the mechanism for relation *Task* 1 *startsA Task* 2. In this figure, *t*1 is a control transition that ensures the simultaneous beginning of both tasks, and *t*2 ensures that *Task* 2 will finish only after the end of the other task. The transitions called *task* 1 and *task* 2 represent the real execution of the tasks. They are modeled by means of transitions with token reservation (represented with the letter "R"), which are non-instantaneous transitions (tokens are removed from their input places when they fire and only some time later are they added to their output places, representing the duration of the tasks' execution).

Another mechanism presented here is the one for managing the resource dependency *sharing N*. The coordination mechanism for this dependency consists of a place *Pn* with *N* tokens representing the available resources. This place is the input place for a transition connecting *request_resource* to *assigned_resource*, defining whether there are available resources. At the end of the task, the token returns to *Pn* via *release_resource*. Fig. 4. shows the model for $N = 3$.

A detailed explanation of the coordination mechanism models has been presented elsewhere [21,27]. The full set of models is available at http://www.dca.fee.unicamp.br/~alberto/pubs/IJCSSE/mechanisms.

The next step is to create software components that implement the modeled coordination mechanisms as

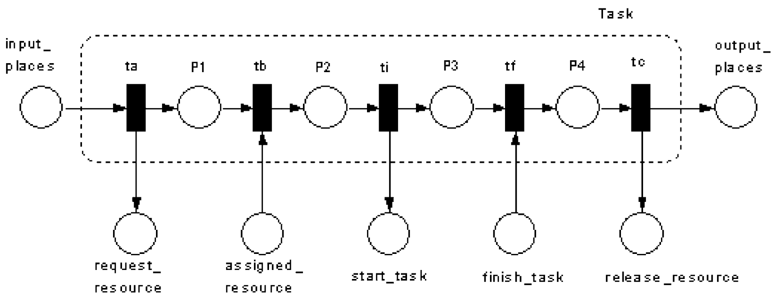

Fig. 2. Task model in the coordination level.

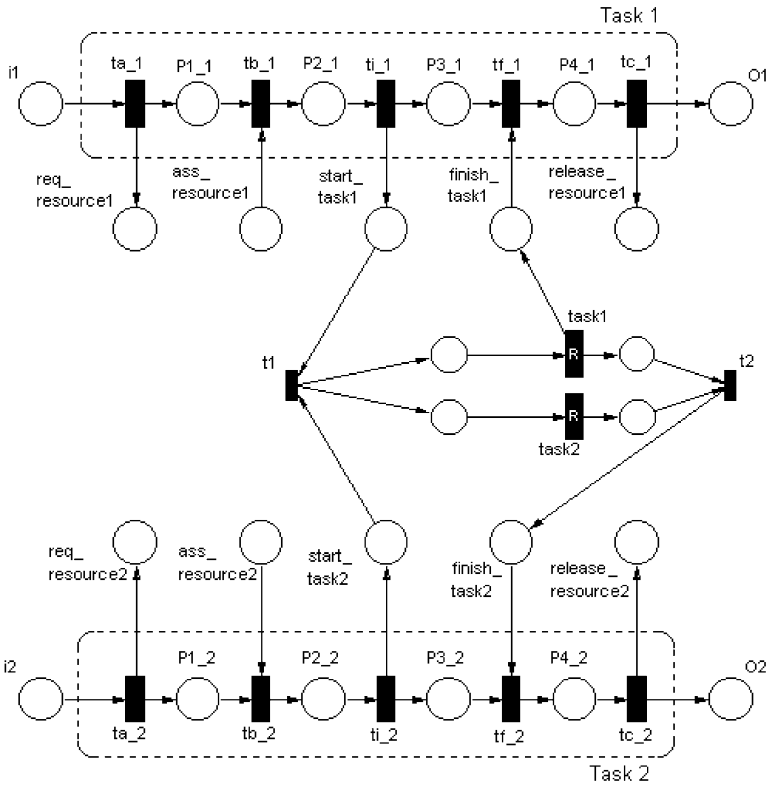

Fig. 3. Model of the coordination mechanism for *Task* 1 *startsA Task* 2.

"black boxes" connecting the collaborative tasks. The coordination components are capable of receiving and generating events from/to tasks in the CVE, controlling their execution. Thus, events related to tasks (e.g., the beginning of a task) may generate output events that affect the execution of interdependent tasks (e.g., blocking the execution of another task).

### 3.3. Coordination components

We have idealized the coordination level as a software layer loosely coupled to the scene. The advantages of this approach are two-fold. It isolates the scene from changes made exclusively on the coordination architecture, and also provides a clear test bench to investigate coordination issues. The list of requirements that should be supported by the loosely coupled coordination layer is as follows:

1. Implementation separated from the scene.
2. Accommodation of changes at the scene, such as removal or inclusion of a task.
3. Changes in the coordination logic should be possible and be restricted to a few software elements.

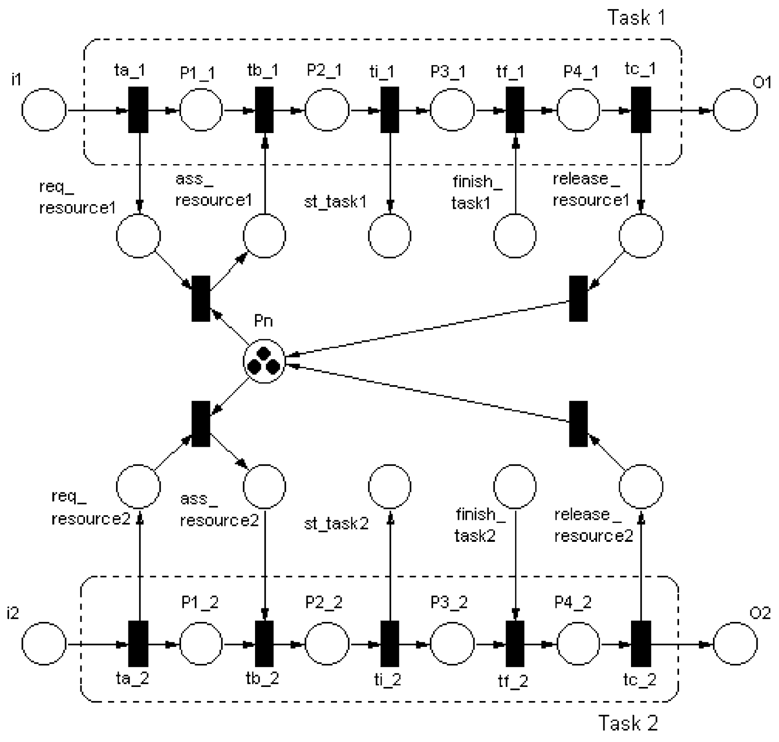

Fig. 4. Model of the coordination mechanism for two tasks sharing three resources.

4. Mixing of different coordination logics within the same coordination control.
5. Attachment of monitoring tools to gather performance related to the coordination control strategy. Potential evaluation indices are throughput, deadlock situations, and load balance.

The architectural decoupling between components and the three-level architecture are solutions for requirements 1–3. The encapsulated implementation of complete services within each component is the solution for requirement 4. The event-oriented communication is the basis for attaching external tools (requirement 5).

### *3.3.1. Coordination level*

Fig. 5 illustrates a high level vision of the components involved in the coordination of interdependencies between two tasks. At the coordination level the figure shows three components, a coordinator and two tasks. The coordinator component implements the coordination mechanisms, both for temporal and resource management dependencies. The task component, instantiated for each task, is responsible for maintaining the task's schedule.

The components of our architecture were designed to communicate by means of the following sequence of events:

*REQUEST START (ReqS)*: When a task is demanded, for example due to a user's action, it must first contact the coordinator to request an authorization for execution. At this time, the coordinator checks if there is any resource management interdependency and, if so, it consults the resource management mechanism to verify if the resource is available (if not, it has to wait until the resource becomes available). Once the resource is available or in the case of having no resource dependency, the coordinator checks if there is any temporal interdependency. If so, it consults the temporal coordination mechanism to verify if the conditions for the task to start have been satisfied (if not, it has to wait until these conditions are satisfied). Once all conditions are satisfied, the signal AutS is sent to the task component.

*AUTHORIZE START (AutS)*: The signal is the authorization given by the coordinator that enables the beginning of a task's execution.

*REQUEST FINISH (ReqF)*: Once the task wants to end its execution, it sends this signal to the coordinator, which verifies if the temporal interdependency (whether it exists) enables the end of the task. If so, it sends the signal AutF to the task and, if there is a resource management dependency, it releases the assigned resource. Otherwise, the co-

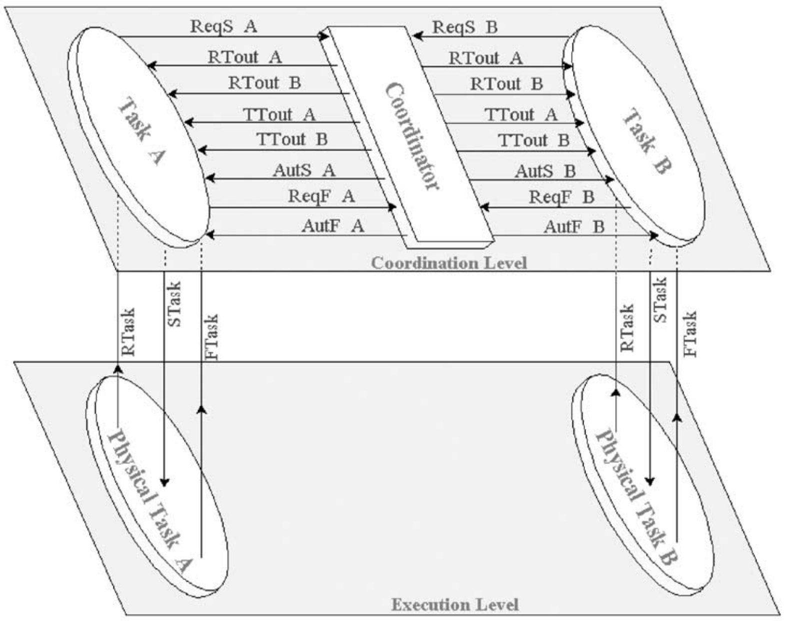

Fig. 5. High level vision of the coordination component between two tasks.

ordinator waits until the temporal coordination mechanism authorizes the end of the task.

*AUTHORIZE FINISH (AutF)*: This signal indicates to the task that it may end.

In the model of the coordinator component, a task has to wait until the temporal and the resource management coordination mechanisms authorize its execution. A possible consequence of this fact is that the task may wait indefinitely if either of these conditions is not satisfied. In order to avoid such situations, the coordinator also sends timeout signals to the tasks.

*RESOURCE TIMEOUT (RTout)*: If the resource is not assigned to the task after a certain waiting time, the coordinator sends this signal.

*TEMPORAL TIMEOUT (TTout)*: If another task does not offer the conditions for the beginning of the task that has requested it, the coordinator sends this signal after a certain waiting time. In this case, if the resource has already been assigned to the task, the coordinator also releases it.

The treatment of timeouts is left to the task components. This gives more flexibility to the architecture, because it does not risk the coordinator component reusability.

The task that had its start unauthorized due to a timeout may execute an alternative task when it receives that signal. This kind of timeout can be thought as an "emergency procedure" to prevent other tasks from being blocked. For example, in a virtual laboratory, the students must follow a sequence of experiments in order to achieve the expected skills. However, if a student cannot conclude a certain experiment after a pre-defined time, a virtual helping agent could be activated in order to guide the user through all the steps of the experiment.

Another possible timeout treatment is to return the task to its initial state. This approach only works when the collaborative activity has an alternative path that avoids the blocked task. In a virtual chat room, for example, if a user cannot start a conversation because there is nobody available, he/she could return to the virtual hall and choose another room.

Another possible consequence of timeouts is that the non-execution of an expected task may invalidate interdependent tasks previously executed. For this reason, the coordinator component sends the timeout signals to all interdependent tasks, and not only to that one which had its execution unauthorized. An example occurs in relation *Task2 afterA Task1*. *Task1* may be executed without restrictions, but in some cases it expects the execution of *Task2* to be "validated". 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 effectuated, after a certain period the order should be canceled. In the example to be presented in the next section, this situation occurs if the monster's wounds are not cauterized after a certain time its head has been severed. In this case, the head is regenerated and the hero must sever it again to kill the monster.

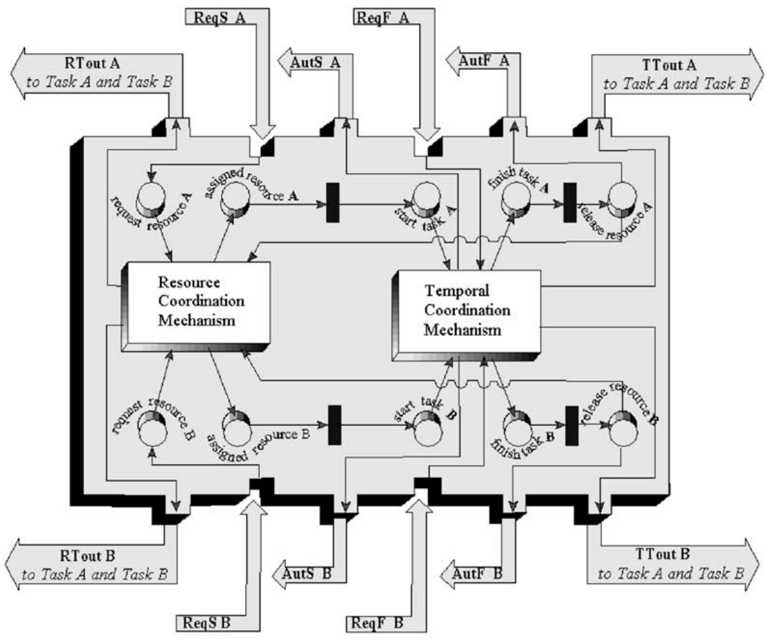

Fig. 6. The coordinator component functioning.

The coordination component encapsulates two PN simulators, one for the temporal and another for the resource management coordination mechanism. Each of these simulators interacts with their associated events, received or sent by the component. Events arriving at the coordinator alter the state of the simulator, while certain states of the simulator may generate output events. Fig. 6 sketches the coordinator functioning.

When the coordinator receives a ReqS signal, it puts a token in the respective request_resource place, starting the resource management mechanism. When the resource is available, the resource management mechanism puts a token in the respective assigned_resource place, which sends it to start_task. The start_task place starts the temporal coordination mechanism, which sends the respective AutS signal indicating the time the task may being. When the temporal mechanism receives the ReqF signal, it checks whether the logical end of the task is authorized and, if so, sends a token to finish_task, which is then passed on to release_resource, indicating to the resource management mechanism that the resource is free again. Finally, the coordinator sends the AutF signal, indicating the logical end of the task.

Although belonging to the same component, each internal coordination mechanism has independent behavior, running a different PN simulator. A natural design choice would have been to consider each mechanism a component itself. However, this would cause a communication overhead between both components when a pair of tasks had both kinds of interdependencies (i.e., temporal and resource management), harming the decoupling characteristic of the components. The choice for considering a component as composed of two coordination mechanisms does not reduce the flexibility of the model, because the component is customizable, i.e., the kind of interdependency to be treated by each of the mechanisms is a parameter of the component. In the particular situation where there is only one kind of interdependency, the other internal mechanism may be considered "empty" with no effects in the coordination of tasks.

### 3.3.2. Execution level

In the previous section we have centered the discussion on the coordination level, where only the logical execution of tasks is considered. The implementation of the coordination components must also consider the execution level, which represents the "physical" execution of the tasks in the CVEs. As shown in Fig. 5, the connection between coordination and execution levels is left to the task components, reducing the responsibilities of the coordinator components, a feature that contributes to their reuse.

The physical tasks communicate with their respective task components by means of the following events:

*REQUEST TASK (RTask)*: This signal is sent from the physical task to the respective component when an event in the CVE (user action, solicitation of another task, elapsed time, etc.) requests its start. The

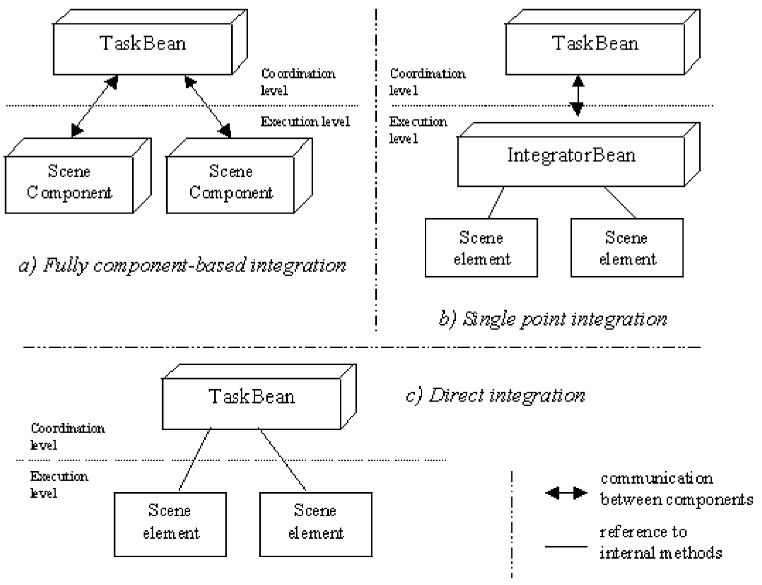

Fig. 7. Approaches for the integration scene-coordination components.

task component forwards this information to the coordinator at the coordination level via ReqS signal.
*START TASK (STask)*: When the coordinator authorizes the start of a task (AutS), the task component forwards this authorization to the physical task via this signal. At this moment, the task really starts its execution in the CVE.
*FINISH TASK (FTask)*: The end of the task's execution in the CVE generates this signal that is sent to the respective component at the coordination level. This information is forwarded to the coordinator via ReqF signal. It is important to clarify that the logical end of the task (i.e., the end from the coordination level point-of-view) occurs only when the task component receives the AutF signal from the coordinator. The logical end of a task guarantees that the temporal interdependencies will not be violated, but may be delayed in relation to its physical end in specific situations.

An advantage of this multiple-level approach is the modularization of the architecture. The coordination model of the CVE may be developed independently from its implementation and vice versa.

Although the only commitment between the coordination and execution levels is the communication by means of the three signals discussed above, the integration issue is not an easy task. This is worsened by the tight relation between scene semantics and its coordination, which directly impacts the communication pattern within both levels. The choice for a communication pattern imposes different kinds of externalization of scene internal elements (i.e., actors, resources, tasks, etc.). In the following we discuss three possibilities of externalization (Fig. 7).

The first possibility is to implement the CVE under the software component paradigm (i.e., scene objects, actors, tasks, and resources implemented as components), which would lead to a straightforward integration with the coordination control. Each task component at the coordination level would be connected to a set of components within the scene. In this approach connections are loosely coupled because the task component (TaskBean—Fig. 7a) does not need to import any scene library/package. This fully component-based integration approach is the ideal one from the software component architecture point-of-view, but it requires a component-based CVE.

A second alternative would be to encapsulate the whole scene into a single IntegratorBean, which would be basically responsible for forwarding coordination commands to the appropriate scene elements and returning scene events to the respective TaskBeans at the coordination level (Fig. 7b). This single point integration approach solution has the advantage of not imposing a reorganization of scene elements, enabling different scene technological implementations. Its drawback is a less decoupled and robust solution, since even a slight change made to the scene would impact the IntergratorBean.

A third approach is to have each TaskBean accessing their respective scene elements (Fig. 7c). The advantage of this direct integration approach is that there is not a unique integration point and, therefore, no need to translate JavaBeans events to method calls in scene elements. Moreover, there is no need to restrict or adapt the scene to the component model. Scene elements may even be implemented as simple structured (non-object-oriented) code. The drawbacks, however, are numerous. The sense of layer is wicked, because coordination components need to import scene libraries. Furthermore, even a single alteration to the scene could impact the coordination components in a less uniform and localized manner than in previous integration approaches.

Since the implementation of a fully component-based CVE is out of the scope of this paper, and our initial goal is to use the coordination approach in existent CVEs without further modifications, we use the direct integration in the videogame implementation presented in the following.

## 4. Prototype implementation: a multi-user videogame

In this section a case study of a CVE is presented, where a user interacts with an autonomous agent that represents a second user. 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 this nephew Iolaus, who cauterizes the monster's wounds after Heracles cuts each head off. However, the last head may not be severed by any weapon. The solution is to bury the monster in a deep hole and cover it with a huge stone.

Fig. 8 illustrates the PN 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 each "actor" (user or agent) may assume either role (Heracles or Iolaus). The upper part of the nets represents Heracles' sequence of tasks and the lower part, Iolaus' sequence of tasks. Heracles must get the sword, sever eight of the Hydra's heads, throw the beast into the hole and cover it with a stone. Iolaus, on the other hand, must get the torch, cauterize the wounds after Heracles has severed the heads and dig the hole.

At the coordination level, the boxes with interdependency names are replaced by the respective coordination mechanism models for simulation and analysis

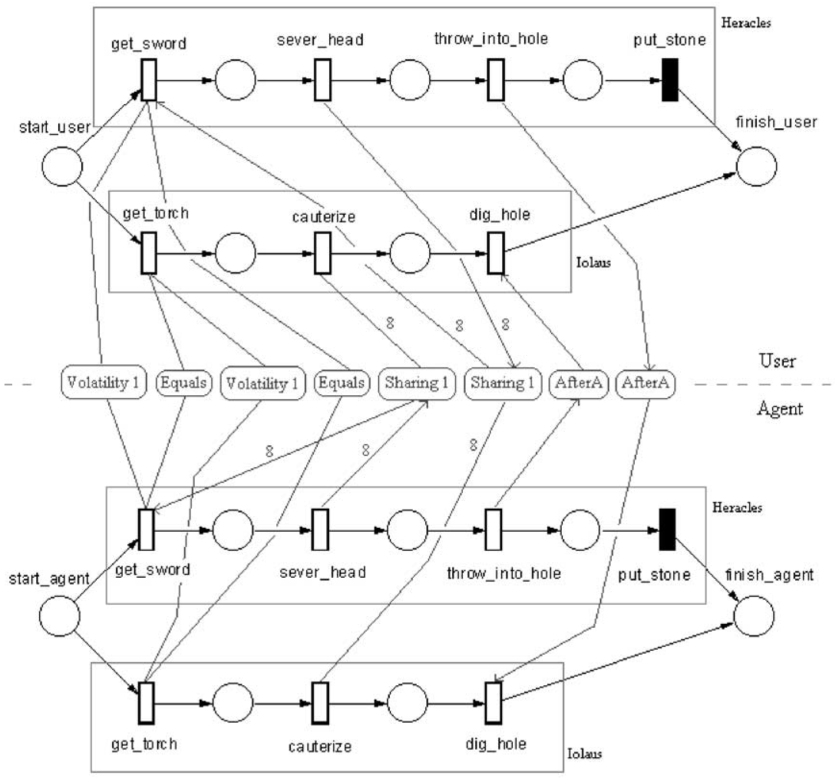

Fig. 8. PN model of Heracles videogame.

purposes. At the execution level, these boxes represent the internal mechanisms of the coordination components to be used.

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 choice of weapon determines the role to be played by each actor. There is also an *equals* interdependency between get_sword and get_torch of different actors. This interdependency forces the agent to choose the other weapon when the user chooses his/hers.

The interdependency *sharing 1* among the tasks get_sword, sever_head and cauterize is the "core" of the game. When Heracles gets the sword, he is also assigned eight resources that may be thought 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. If the head wound is not cauterized within a certain period after it has occurred, the timeout signal sent by the coordination mechanism causes the task to return to its initial state and also reassigns the resource to Heracles, indicating that he must once again sever that head (now regenerated).

There is also an interdependency *afterA*, indicating that Heracles may only throw the monster in the hole if Iolaus has already dug it.

The PN model was simulated and analyzed with the aid of a developed tool [21] (actually, the model shown in Fig. 8 is an enhancement of previous models whose simulation has detected problems).

The videogame was implemented by using the Blaxxun Contact [29], a client for multimedia communication that provides resources for virtual reality modeling language (VRML) visualization, chats, message boards, avatars, etc. In this implementation we have used the VRML visualization and the avatars with predefined movements.

The interaction with the user occurs by means of buttons defined in a Java applet that interacts with the VRML world via external authoring interface (EAI), 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 (the physical execution of the tasks is decoupled from their coordination). At the implementation, task components are connected to the interface's buttons, enabling or disabling them whether their respective tasks are enabled or not. The integration of the components with the VRML scene is direct (Fig. 7c) because the components call methods of the applet graphical interface elements, and vice versa. The applet, however, could be redesigned as an integrator bean (Fig. 7b), which would enhance the decoupling of the architecture.

In order to give more dynamism to the game, the agent has an aleatory behavior, taking a variable time to start 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 cut by Heracles before it is reborn. Fig. 9 shows some frames of the videogame (the nine heads of Hydra are represented by nine monsters). Frames *a* and *b* show Heracles' interface, while frames *c* and *d* show Iolaus' interface.

## 5. Conclusion

This paper has presented an approach for behavior coordination in CVEs, which is a step towards shortening the gap between virtual environments and CSCW practices regarding the carrying-out of tightly coupled collaborative activities. The approach started with the basic idea of defining a set of interdependencies that frequently occur between collaborative tasks. For each interdependency, a coordination mechanism was formally defined as a PN model. At this level, the whole environment may be expanded as a PN for simulation and analysis, enabling the anticipation of possible problems. Finally, we have also developed a component-based architecture to implement the coordination control in CVEs in a modular and pluggable way.

Our coordination approach follows a three-abstraction levels hierarchy, which has the advantage of isolating the parts of coordination design. For instance, if the CVE designer wants to implement a videogame such as the one presented in Section 4, it is not necessary to consider the PN model of the system. Designers may simply use predefined coordination components, which encapsulate the PN logic of the coordination mechanisms, hiding this logic from designers. Thus, it is not necessary to attach a PN model to a CVE. On the other hand, in initial phases of the design, only the model may be used, dismissing implementation details.

Another important aspect of the presented coordination approach is that is treats the CVE's behavior independently from its form and function. In other words, the behavior does not depend on animation and modeling techniques. For example, the videogame presented in Section 4 may be redesigned to include more sophisticated scenario, avatars and movements without affecting the coordination infrastructure. Moreover, a whole task execution may be redefined without affecting the global behavior (for instance, instead of using the sword to sever the heads, Heracles could get a gun to shoot them).

As future research, we plan to formalize the presented approach (from the coordination logic to components implementation) in order to create a generic coordination framework that can be used not only in CVEs, but also in other kinds of collaborative applications.

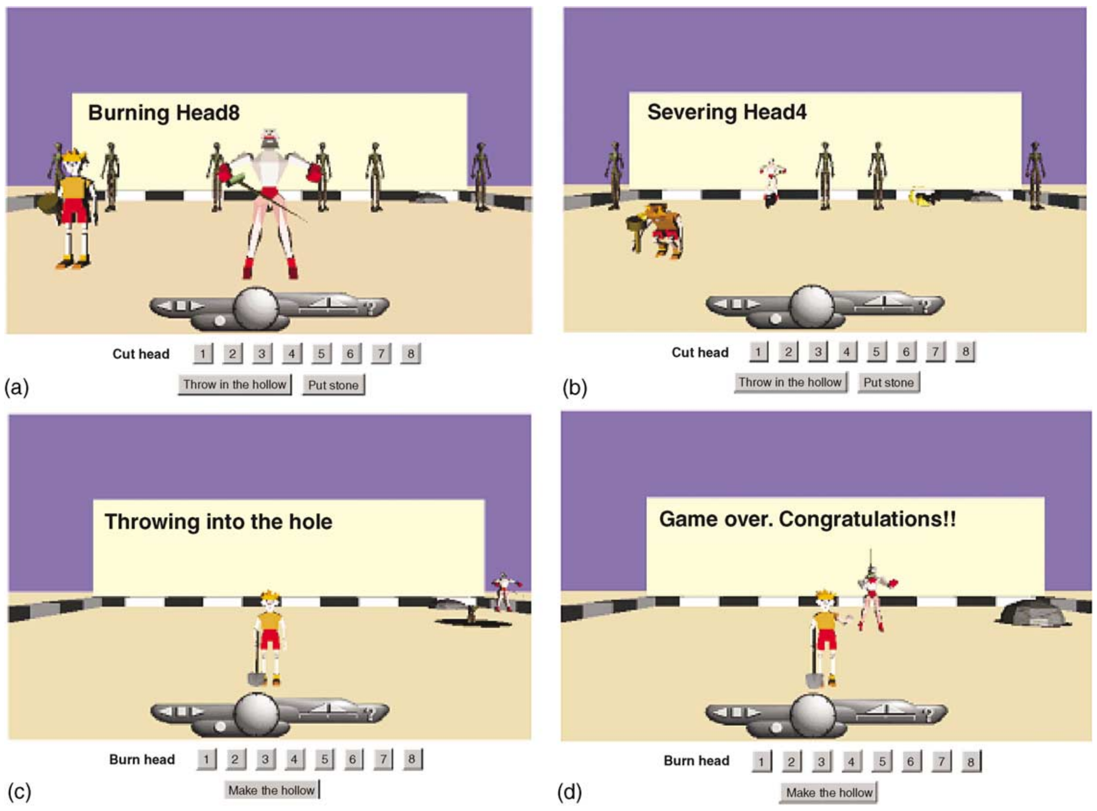

Fig. 9. Frames of Heracles videogame.

Concerning the CVE application, we have developed here a basis for further research on the assumption that virtual environment behavior reflects collaboration patterns that are the outcome of coordination control strategies. Isolating the three dimensions is a fundamental step to understand how they intertwine in CVE. Following this reasoning, we suggest that an ample research on infrastructure for CVE is already necessary. The coordination apparatus demonstrated here is one of the many aspects of such complex framework.

Finally, we would like to reinforce our belief that the upcoming virtual society will be strongly based on CSCW and net-VE technologies. For this reason, research ventures towards shortening the gaps between both technologies, such as that of coordination focused here, are important steps towards the settlement of this society.

## Acknowledgements

The first author is sponsored by FAPESP (Foundation for Research Support of the State of São Paulo, Brazil), grant number 00/10247-3. The second author is sponsored by CAPES/PICD. Thanks also to DCA/FEEC/Unicamp and CEUD/UFMS for the expressive support granted to this research.

## References

[1] Igbaria M. The driving forces in the virtual society. Communications of the ACM 1999;42(12):64–70.
[2] Bannon LJ, Schmidt K. CSCW: Four characters in search of a context. In: Bowers JM, Benford SD, editors. Studies in Computer Supported Cooperative Work. Amsterdam: North-Holland, 1991. p. 3–16.
[3] Hagsand O. Interactive multiuser VEs in the DIVE system. IEEE Multimedia 1996;3(1):30–9.
[4] Singhal S, Zyda M. Networked virtual environments: design and implementation. Reading, MA: Addison-Wesley, 1999.
[5] Benford S, Bowers J, Fahlen L, Mariani J, Rodden T. Supporting cooperative work in virtual environments. The Computer Journal 1994;37(8):653–68.
[6] Gutwin C, Greenberg S. Workspace awareness for groupware. Proceedings ACM Conference on Human Factors in Computing Systems (CHI), Vancouver, BC, Canada, 1996. p. 208–9.
[7] Rodden T. Populating the application: a model of awareness for cooperative applications. Proceedings

ACM Conference on Computer-Supported Cooperative Work (CSCW), Boston, MA, USA, 1996. p. 87–96.

[8] Frécon E, Nöu AA. Building distributed virtual environments to support collaborative work. Proceedings ACM Symposium on Virtual Reality Software and Technology (VRST), Taipei, Taiwan, 1998. p. 105–19.

[9] Perry TS. In search of the future of air traffic control. IEEE Spectrum 1997;34(8):18–35.

[10] Malone TW, Crowston K. What is coordination theory and how can it help design cooperative work systems? Proceedings ACM Conference on Computer-Supported Cooperative Work (CSCW), Los Angeles, CA, USA, 1990. p. 357–70.

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

[12] Hindmarsh J, Fraser M, Heath C, Benford S, Greenhalgh C. Object-focused interaction in collaborative virtual environments. ACM Transactions on Computer–Human Interaction 2000;7(4):477–509.

[13] Joslin C, Molet T, Magnenat-Thalmann N, Esmerado J, Thalmann D, Palmer I, et al. Sharing attractions on the net with VPark. IEEE Computer Graphics and Applications 2001;21(1):61–71.

[14] Kim GJ, Kang KC, Kim H, Lee J. Software engineering of virtual worlds. Proceedings ACM Symposium on Virtual Reality Software and Technology (VRST), Taipei, Taiwan, 1998. p. 131–8.

[15] Attie PC, Singh MP, Emerson E, Sheth A, Rusinkiewicz M. Scheduling workflows by enforcing intertask dependencies. Distributed Systems Engineering Journal 1996; 3(4):222–38.

[16] McIlroy MD. Mass produced software components. In: Naur P, Randell B, editors. Software Engineering. Report on a conference sponsored by the NATO Science Committee, Scientific Affairs Division, NATO-Brussels, 1968. p. 138–55.

[17] Szyperski C. Component software: beyond object-oriented programming. Reading, MA: Addison-Wesley, 1998.

[18] Garlan D, Shaw M. Introduction to software architecture. In: Ambriola V, Tortola G, editors. Advances in Software Engineering and Knowledge Engineering, vol. I. Singapore: World Scientific Publishing Co., 1993.

[19] Sun Microsystems. JavaBeans. ⟨http://java.sun.com/products/javabeans⟩, May 2001.

[20] Roussev V, Dewan P, Jain V. Composable collaboration infrastructures based on programming patterns. Proceedings ACM Conference on Computer Supported Cooperative Work (CSCW), Philadelphia, PA, USA, 2000. p. 117–26.

[21] Raposo AB, Magalhães LP, Ricarte ILM. Petri Nets based coordination mechanisms for multi-workflow environments. International Journal of Computer Systems Science & Engineering 2000;15(5):315–26. Special Issue on Flexible Workflow Technology Driving the Networked Economy.

[22] Mascarenhas R, Karumuri D, Buy U, Kenyon R. Modeling and analysis of a virtual reality system with time Petri Nets. Proceedings International Conference on Software Engineering (ICSE), Kyoto, Japan, 1998. p. 33–42.

[23] Zhou Y, Murata T, DeFanti T, Zhang H. Fuzzy-timing Petri Net modeling and simulation of a networked virtual environment—NICE, IEICE Transactions on Fundamentals of Electronics, Communication and Computer Science, PART A, 2000;E83-A(11):2166–76.

[24] Magalhães LP. Raposo AB, Ricarte ILM. Animation modeling with Petri Nets. Computers & Graphics 1998;22(6):735–43.

[25] Ellis CA, Wainer J. A conceptual model of groupware. Proceedings ACM Conference on Computer Supported Cooperative Work (CSCW), Chapel Hill, NC, USA, 1994. p. 79–88.

[26] Allen JF. Towards a general theory of action and time. Artificial Intelligence 1984;23:123–54.

[27] Raposo AB. Coordination in collaborative environments using Petri Nets. Doctorate thesis, DCA—FEEC—UNICAMP, October 2000 [in Portuguese].

[28] van der Aalst WMP, van Hee KM, Houben GJ. Modelling and analysing workflow using a Petri-net based approach. Proceedings of the Second Workshop on Computer-Supported Cooperative Work, Petri Nets and Related Formalisms, Zaragoza, Spain, 1994. p. 31–50.

[29] blaxxun interactive. blaxxun Contact 4.4. ⟨http://www.blaxxun.com/products/contact⟩, August 2000.