CAHIER DU LAMSADE

Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la Décision
(Université Paris-Dauphine)
Unité de Recherche Associée au CNRS ESA 7024

A FRAMEWORK INTEGRATING COLLABORATION AND COORDINATION

CAHIER N° 179
avril 2001

Marie-José BLIN [1]
Françoise FABRET [2]

received: October 2000.

[1] *LAMSADE, Universié Paris-Dauphine, Place du Maréchal De Lattre de Tassigny, 75775 Paris Cedex 16, France (blin@lamsade.dauphine.fr).*

[2] *INRIA, 78153 Le Chesnay, France (Francoise.Fabret@inria.fr).*

CONTENTS

Pages

Résumé 1
Abstract 1

1. Introduction 2

2. Problem formulation 4
2.1 Context of a complex project specification 4
2.2 Task view 4
2.3 Problem definition 6

3. The framework 7
3.1 The cooperation architecture 8
3.2 Cooperation specification 9
3.2.1 Collaboration type 10
3.2.2 Coordination protocol 12
3.3 Protocol execution model 14

4. Related work 15

5. Conclusion 16

References 17

Un modèle de travail coopératif supportant les concepts de collaboration et de coordination

Résumé

Dans le domaine du travail coopératif, deux grands axes de recherche ont été explorés : celui des workflow et celui du travail en groupe. Le premier axe se concentre sur la coordination de tâches et le second sur la mise en place d'un environnement de travail partagé permettant la collaboration. Dans les application réelles impliquant le travail coopératif d'un grand nombre d'équipes, les besoins en coordination et en collaboration sont étroitement liés. Les nombreux modèles de worflow existants n'intègrent pas le concept de collaboration. Nous proposons, dans ce papier, un modèle considérant les deux dimensions, à la fois, celle de la coordination et celle de la collaboration. Notre proposition est illustrée par une application réelle de spécification de systèmes orientée réutilisation.

Mots clés : travail coopératif, collaboration, coordination, modèle

A Framework Integrating Collaboration and Coordination

Abstract

In the cooperative work area, two main directions have been explored: workflow and groupware. The former focuses on the task coordination and the later on common work environment, i.e., collaboration between partners. In real world applications involving a lot of teams, both coordination and collaboration concepts are closely linked. Actually, modeling workflow is well overcame, nevertheless the integration of collaboration in workflows is an open issue. In this paper, we propose a framework which integrates collaboration and coordination dimensions. Our proposal relies on a real large-scale application intended to specify complex distribution management systems by reusing product family components.

Key words: cooperative work, collaboration, coordination, framework

1 Introduction

The specification process of large-scale projects (or systems) requires the collaboration of a lot of teams with different skills working in a common extent. Specification work is too long and too complex to be executed as a single task, so the specification process is decomposed into several specification tasks. The decomposition is motivated by different abstraction levels of decision which constitute a decision graph. Consequently, the specification phase can be represented by a specification graph where each node corresponds to a specification task.

A specification task is in charge of specifying a certain set of elements with respect to decisions derived from the specification produced by upper level tasks. Several teams may participate to one specification task. In this case, each of them is in charge of a subset of elements, and the teams have to collaborate in order to ensure the consistency of the final specification produced by the task. So, the specification phase of a project can be seen as a workflow where each specification task gathers all the teams who have to collaborate to elaborate the specification the task is in charge of.

Take for example a concrete project devoted to distribution management. The final goal is to build a specific distribution management system for a particular store which may be a small supermarket as well as an hypermarket. The system will be installed in a specific country with specific management rules, and of course a specific language. Due to requirements of the customer, the system may have to respect specific standards and specific management procedures, and to provide particular functionality. Of course, it has to be integrated to the customer technical environment. Specifying such a system requires several decision levels as shown in Figure 1. From a general description of the customer's store, and of the needs, the general characteristics of the future system are defined by the top level specification task. These characteristics are used in two independent specification tasks at the second level: one defines the general functional specification of the system and its technical environment, and the other, the organization of the system installation. At the final level, we find several tasks, among them the tasks which specify all the hardware and software elements composing the system, the tasks specifying the documentation, and the tasks defining the relevant customer services. The tasks are represented by squares in Figure 1. Bullets in a square represent teams, and dashed arrow, the collaboration links between them. Finally, solid arrows connecting the squares represent precedence relationships between the tasks. One task cannot begin while all its predecessors have not produce the decisions used by the task. For simplification reasons, Figure 1 does not show all the specification tasks. For example, generally, the set of hardware and software elements is composed of several independent sub-sets of elements. By independent sub-sets, we mean parts of the system that may be specified independently from each other without risk of mutual inconsistency. To allow simultaneous work, independent sub-sets are specified in independent tasks.

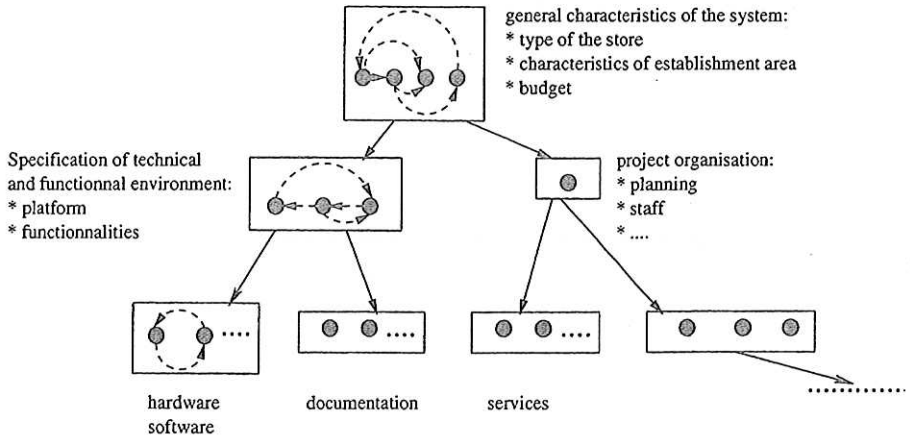

Figure 1: The top-down specification tasks of a distribution management system

In this paper, we are interested by organizing the teams work during one specification task (for example the hardware software specification task of Figure 1). Such a task has to provide a complete and consistent specification. Generally the teams involved in a specification task work as follows: given a set of elements to be specified, organized in several disjoint subsets, each of them is specified by a specific team who produces the specification of each element of the subset it is in charge of, so that all the specified elements of the task are mutually consistent. Remark that in such a work organization, the work is not shared at the team level. Nevertheless, as the work of each team may impact the work of the other ones, the work of the different teams have to be coordinate, and the necessary collaboration between the teams (for example, exchanging opinions about possible choices) has to be organized. We claim that it is possible to control collaboration and coordination of several teams in an integrated workflow as long as the control is based on a well-suited architecture.

We propose a framework for declaratively specifying a workflow integrating coordination and collaboration. The framework consists of three components. The first component is a cooperation architecture, the second one is a language for specifying collaboration forms, like, for example, iterative conversations between participants, discussions where each participant alternately develops its propositions and the different propositions are communicated to all the group, negotiation [Munier, Shakun, 1988]. Finally, the third component is a language for specifying a coordination protocol which stipulates and mediates the articulation of the collaborative work. Changes to the state of the protocol induced by one actor (a person or a process) are conveyed to the other actors. The execution flow of the protocol may be disturbed by user interventions like stopping, restarting or iterating an activity.

The remainder of the paper is organized as follows. Section 2 formulates the problem and provides an example inspired of a real large-scale project which is used throughout the paper to illustrate the different notions and solutions. Section 3 is the corner stone of the paper, it describes the framework. Section 4 exposes some existing work in modeling workflows. Finally, section 5 concludes and presents future work.

2 Problem Formulation

In this section, the general context of a complex project specification is given. Then, the support for each specification task is presented. Finally, the problem considered in this paper is precisely defined.

2.1 Context of a complex project specification

Actually, enterprises tend to reuse their knowledge and know-how as much as possible. So, they constitute a "corporate memory" that describes experiments, and work done in the past, plus lessons from them. The corporate memory contains design options that are available for a system related to high-level user requirements and the effects of new design proposals on the overall system. It is used for specifying new projects, in particular, part of existing specifications may be reused. In this paper, we suppose that a knowledge base containing all existing specifications is used as the support of the specification of any new project. In particular, the knowledge base contains information over compatibilities and incompatibilities between object versions which have been established by previous experiments, or by any external knowledge.

We explained in the previous section that a specification phase is composed of several specification tasks (simply called tasks for short) and that the decomposition is motivated by the different levels of decision. A given task is in charge of specifying a certain set of interdependent objects with respect to decisions produced by upper level tasks. In turn, it produces as result, a set of decisions about the objects it has to specify.

2.2 Task view

The support of a specification task is a view of the global knowledge base concerning the objects to be specified by the task, where versions being contradicting with upper level decisions have been discarded. It is commonly admitted that it is necessary to take into account the product evolution due to the long span of life of the products and to their reusability feature. So, it seems realistic to explicitly give the possibility of creating a new version for certain objects in a specification task view.

In the following, we propose an example of a task view organization which fits with the requirements above. Let us remark that our framework does not depend on any particular form of task view organization. Our goal in presenting a precise organization for the task view is to provide a well understandable application example, and thus clearly illustrate our proposal. The view for a given specification task is a collection of consistent configuration descriptions (simply called descriptions for short in the remaining). Each of them provides three information for each object specified in the task: positive compatibility, evolution possibility and recommendation about the presence of the object.

A positive compatibility for an object is a set of compatible existing versions of the object. The set of positive compatibilities of a description represents consistent configurations having successfully tested and validated in previous projects. In other words, if, for each object of the task, a version is chosen among the versions provided by its positive compatibility, the resulting configuration will be consistent.

Evolution possibility for an object allows or forbids the building of an innovative configuration by the creation of a new version of the object or by the use of a version which is not included in the positive compatibility of the object. It is composed of:

1. a novelty recommendation, e.g., no novelty, strong novelty recommendation,
2. in the case where novelty is authorized (or encouraged), additional information intended to avoid identified inconsistency risks. This information consists of a set of forbidden versions, i.e., versions which will surely lead to inconsistent configurations.

Recommendation about the presence of an object indicates if the object is mandatory or optional. More refined values may exist, e.g., possible but not recommended, absence or indifferent absence, or presence.

Thus, each description of a task view is so that:

1. if, for each object of the task, a version is chosen among positive compatibility of the object, the resulting configuration will be surely consistent,
2. if, for one object which has a novelty authorization, a version is chosen among the forbidden ones or a new version is created, the resulting configuration will be able to be inconsistent.

Example of a task view. Let us come back to example of Figure 1, and consider the task which is in charge of specifying the cash sub-system (such a task is one of the hardware-software specification tasks). This task will be used to illustrate the different notions presented all along the remaining of this paper.

A cash subsystem is composed of a scanner, a check reader, a card reader, a keyboard, and a program named TCF. The existing versions of each of these objects are those defined in Table 1. The view for the cash sub-system specification task is presented in Table 2. It consists of two descriptions d and d'.

Let us explain the first description, d. It says that:

1. The compatibility between versions AU33_SD Board 3 and AU32_N Board 3 of the program TCF, version PEACH of the scanner, version DASSAULT of the check reader, versions GEMPSY 9600 bauds and ICL 1200 bauds of the card reader and version FPI of the keyboard is successfully tested.
2. Except for the keyboard, specifying every object is mandatory.
3. No evolution possibility is given for the specification of the scanner, the card reader and the keyboard. On the opposite, it is possible to create new versions of the program TCF, i.e., versions other than AU33_SD Board 3 and AU32_N Board 3. Nevertheless version AU33_SD Board 3 cannot be chosen (that may be due to previous experiments which revealed incompatibilities between this program version and the other object versions in d). Creating a new version for the program is encouraged and it is the same for the check reader.

object	versions
program TCF	AU33_SD Board 3 AU33_ND Board 3 AU32_N Board 3
scanner	PEACH NCR
check reader	DASSAULT ICL IBM
card reader	GEMPSY 9600 bauds ICL 1200 bauds DASSAULT 1200 bauds
keyboard	FPI

Table 1 : Objects and object versions of the cash sub-system specification task

2.3 Problem definition

In this paper we address the problem of organizing the collaboration between teams participating in a specification task so that:

1. the resulting decisions provide a consistent specification of the objects, and
2. the work needed to reach a consensus is minimized in terms of time and negotiation efforts.

More precisely, we propose a framework for specifying an extended worflow from constraints (the task view), team working modes, and specific domain application knowledge. By extended workflow we mean a worflow which integrates collaboration and integration Due to frequent changes in the knowledge over the constraints (for example technical evolution leading to new object versions), over the teams (e.g., team merging or splitting), the framework have to provide facilities:

1. to declaratively define the different parameters entering in the workflow,
2. to reuse part of existing workflow (i.e., a worflow actually in the knowledge base).

Our framework consists of a cooperation architecture, a model for defining team working modes, and a workflow specification language. The components of the framework are described in the following section.

desc.		definition
d	*d_program:*	{object : program TCF, presence : mandatory positive compatibility : AU33_SD Board 3, AU32_N Board 3 evolution possibility : encouraged novelty (forbidden : AU33_ND Board 3)}
	d_scanner:	{object : scanner, presence : mandatory positive compatibility : PEACH evolution possibility : no novelty}
	d_check reader:	{object : check reader, presence : mandatory positive compatibility : DASSAULT evolution possibility : strong novelty recommendation (forbidden : ICL)}
	d_card reader:	{object : card reader, presence : mandatory positive compatibility : GEMPSY 9600 bauds, ICL 1200 bauds evolution possibility : no novelty}
	d_keyboard:	{object : keyboard, presence : optional positive compatibility : FPI evolution possibility : no novelty}
d' =	*d'_program:*	{object : program TCF, presence : mandatory positive compatibility : AU33_SD Board 3, AU33_ND Board 3 evolution possibility : encouraged novelty (forbidden : AU32_N Board 3)}
	d'_scanner:	{object : scanner, presence : mandatory positive compatibility : NCR evolution possibility : no novelty}
	d'_check reader:	{object : check reader, presence : mandatory positive compatibility : ICL, DASSAULT evolution possibility : strong novelty recommendation (forbidden : IBM)}
	d'_card reader:	{object : card reader, presence : mandatory positive compatibility : ICL 1200 bauds, DASSAULT 1200 bauds evolution possibility : no novelty)}
	d'_keyboard:	{object : keyboard, presence : mandatory positive compatibility : FPI evolution possibility : no novelty)}

Table 2 : The cash sub-system specification task view

3 The Framework

First, the cooperation architecture is described, then the team working modes model and the workflow specification language are presented. Finally, workflow execution model is depiected.

3.1 The cooperation architecture

In what follows, a team is an agent (possibly human) in charge of an object or of a group of objects specified together. Of course, a team may be actually a group of agents, which will functionally be considered as only one.

Teams do not directly communicate with each other and a team does not necessarily know the other teams. A coordinator is in charge of the teams coordination and drives the work sequencing among the teams by sending messages to them. In the same way, the teams report the result of their work by messages sent to the coordinator.

The cooperation semantics is described by (1) the knowledge used by the coordinator to elaborate the messages sent to the teams, (2) the knowledge conveyed by the messages, (3) the coordinator execution model.

The cooperation model is based on a fixed architecture shown in Figure 2. The corner stone of the architecture is a coordinator having a fixed execution model. It uses a dynamic and a static knowledge. The first is fixed, while the other is the medium of the collaboration specification for a given application. The static knowledge consists of (1) a task view as defined in section 2, (2) the team definition, i.e., the mapping between the objects of the task view and the teams, (3) the definition of the collaboration types, (4) the coordination protocol that is first the work steps definition, each work step specifying how the teams collaborate by associating a collaboration type with a set of teams, and second the control of the work steps: static scheduling together with exception specifications in the form of external or internal events. Points (3) and (4) are described in detail later in this section.

The dynamic knowledge consists of (1)the history of the work evolution, i.e., the history of the messages exchanged with the teams, plus the exception event occurences, and (2) the current state of the work execution.

A message sent by the coordinator to a team contains information about the work to do: (1) the concerned objects, (2) the collaboration type to respect, (3) a subpart of the task view that is relevant for the objects, and (4) possibly some knowledge over the work evolution that is useful for the team work.

The main feature of this architecture is that only the coordinator has a global knowledge over the cooperation (at every moment it knows the history and the plan for the future steps), while the team knowledge is local and instantaneaous (a team has no way to anticipate the future).

The coordinator controls the cooperative work as follows. Every time an exception event or a message from a team arises, this event (or message) is registered in the history. Then, the coordinator executes the following algorithm:

```
on receive message from a team or an exception event
  (C, T) := compute_next_step();
  for t in T do
    m := compute_message(C,t);
    send_message(m,t);
```

```
od;
```

The compute_next_step function computes a collaboration type C and a set of teams T. To do that it uses both the static knowledge plus the current state of the work and of the history. As a side effect, the function updates the current state of the work. The compute_message function uses the collaboration type C to compute the message to send to each team in T.

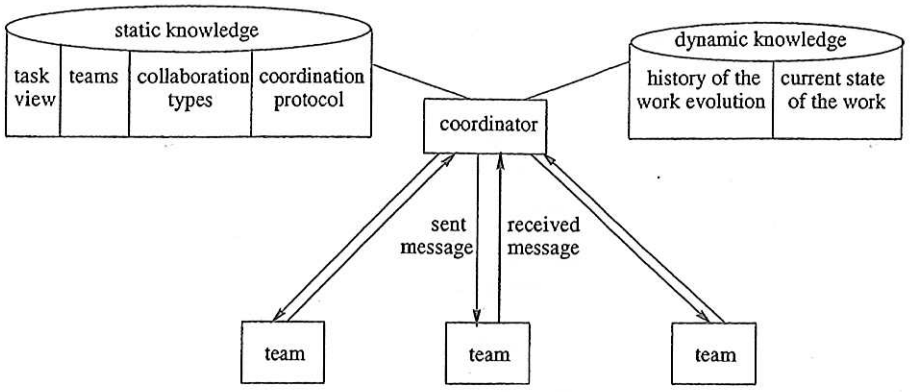

Figure 2: Collaboration architecture

3.2 Cooperation Specification

In our model, we consider that specifying a cooperation requires to define a workflow, and also collaboration between teams. For example, suppose that two teams A and B cooperate for specifying objects a and b, respectively. During a first step, the teams discuss and exchange opinions over the versions to choose for the objects, then a decision step starts where A (resp. B) chooses one version of a (resp. a version of b). The workflow for this scenario is depicted in Figure 3, it contains two work steps: discussion and decision. Each of theses steps is a sub-workflow. The discussion sub-workflow consists of two activities: "A discusses about a", and "B discusses about b", these activities may be iterated. Similarly the decision workflow consists of two activities: "A decides about a", and "B decides about b", there is no iteration in the decision step, first B decides and then A. To be precise the worflow specification has to include the number of discussion iterations and the exception events for both steps. What is not appearant in the workflow is the activity description. What is the meaning of "A discusses about a" and "B discusses about b"? For example what is the action of A and B:

- may a team produce several propositions?
- how can a team give its preferences between different propositions?
- has a team to consider the propositions made by the other team during previous iterations?

- may a team raise an exception in order to interrupt the work step?

The responses for A are valid for B, they determine the semantics of the discussion between A and B. Quite similar questions arise for the meaning of "A decides about a", and "B decides about b". Even the questions are the same, the responses are probably different depending on the fact that A and B are discussing or deciding. In our example, we need to provide responses concerning the discussion and those concerning the decision.

In our model we specify the collaboration via the concept of *collaboration type*, and the coordination via what we call a *coordination protocol.*

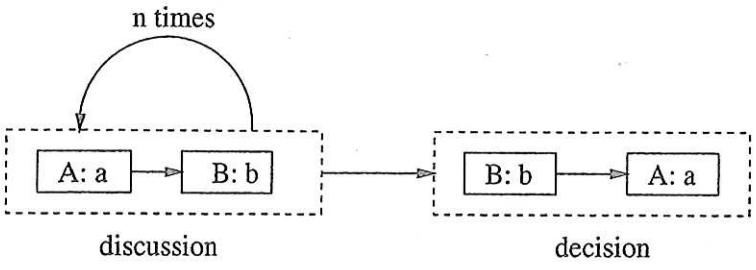

Figure 3: An example of cooperation

3.2.1 Collaboration type

A given collaboration form between teams implies that all the teams have a certain behaviour in common. So defining a collaboration form comes to describing this common behaviour. As, in our model, the artifact of the collaboration consists of messages exchanged by the coordinator and the teams, the way of specifying the required common behaviour is to declaratively specify what knowledge has to be conveyed by the messages.

A *collaboration type* is an abstract data type modelling a collaboration form. This type is a nuplet < id, history synthesis, collaboration recommendation, task view filter, team_to_coordinator, exception> where:

- *id* is the name of the collaboration type (this name is unique, two different collaboration type have different names),
- *history synthesis* is a function that computes a synthesis of the collaboration evolution from a work evolution history.
- *collaboration recommendation* describes the required behaviour of teams with regard to the collaboration evolution.
- *task view filter* is a function extracting a part of the task view from a work evolution history and a task view.
- *team_to_coordinator* describes the information that has to be put in the responses of the teams.
- *exception* is a set of events associated with particular states of the work evolution history.

An example of collaboration type: "controlled discussion" The controlled discussion may be used when partners having to collaborate for specifying a set of objects need to exchange their opinions. The teams are invited to strongly collaborate (i.e., each team has to consider as much as possible the choices of the other teams), nevertheless each of them has the possibility to veto solutions proposed by the other teams. The result of a controlled discussion may range a spectrum from a convergence of the opinions toward one (or several) solution(s), to a total divergence (the teams veto all the possible solutions). So a controlled discussion type provides two exception events: "convergence" and "divergence". When such events occur, there is no matter to go on the discussion.

This collaboration type may be used in the application presented in Section 2, it is also often used to elaborate teacher plannings.

A controlled discussion consists of three phases. During an initial phase, the partners are informed of the collaboration recommendation, then a set of possible specifications is gradually built during the second phase, finally the partners agree to set up specifications preference criteria.

Phases 1 and 2 of the collaboration type applied to the example of Section 2 are as follows:

phase 1: a set of consistent configurations is gradually built. To do that one team elaborates a proposition by giving weighted versions for the object it is in charge of. Then another team elaborates its proposition by giving weights to versions of its objects that are consistent with the choices made by the first team (i.e., it can only choose versions which match with at least one of the versions mentionned by the first team), and so on for the remaining teams (each of them taking into account the propositions of the preceding teams).

phase 2: starting from the set of weighted configurations produced in the first phase, the goal of this phase is to find an agreement over the weights.
The normal discussion processus described above may be aborted by the occurrence of a convergence or divergence event.

This behaviour is achieved by the components of the collaboration type specification. In order to illustrate how each of these components work, we give the step-by-step execution of a discussion beetwen two teams A and B as represented by the discussion work step of Figure 3. Teams A and B are respectively in charge of scanner and card reader objects specified by the task view of Table 2. The execution steps for phases 1 and 2 are shown in Table 3.

	p	H	T	HS	task view filter		team_to_coordinator	
1	1	$\emptyset$	A	$\emptyset$	$s_{A,1}$:	d_scanner d'_scanner	$r_{A,1}$:	d_scanner, 8 d'_scanner, 6
2	1	$s_{A,1}$ $r_{A,1}$	B	$r_{A,1}$	$s_{B,2}$:	(d_scanner,8), d_card reader (d'_scanner,6), d'_card reader	$r_{B,2}$:	d_card reader, 5 d'_card reader, 9
3	2	$s_{A,1}$ $r_{A,1}$ $s_{B,2}$ $r_{B,2}$	A	$r_{A,1}$ $\&r_{B,2}$	$s_{A,3}$:	$r_{A,1}\&r_{B,2}$	$r_{A,3}$:	d_scanner,7 d'_scanner,7
4	2	$s_{A,1}$ $r_{A,1}$ $s_{B,2}$ $r_{B,2}$ $s_{A,3}$ $r_{A,3}$	B	$r_{B,2}$ $\&r_{A,3}$		raises convergence		

Table 3: Controlled discussion execution

The discussion runs in four execution steps (column 1). P (column 2) indicates the collaboration phase, H is the current state of the working evolution history (history for short), it consists of all the messages sent ($s_{team,exec\_step}$ messages) to the teams and returned by them ($r_{team,exec\_step}$ messages). T is a team. HS is the result returned by the history synthesis function, this result consists of a concatenation of messages extracted from H. The task view filter column gives the result returned by the associated function, namely a message to send to a team, it may raise the exception event "convergence" or "divergence". The last column of the table is the message returned by the teams. Such a message consists of a set of weighted object descriptions.

The content of the table reflects the following policies:

- history synthesis: during phase 1, the fuction returns the team-to-coordinator messages contained in the history. During phase 2, it returns the most recent message returned by each team.
- task view filter: takes in input the result P computed by the History synthesis function, and a team T. During phase 1, it uses the task view to complete each configuration described in P with a consistent description of objects of T. During phase 2, the function tests P to detect if there is still reason to go on with the discussion because there is no more choice problem to solve. If that is the case, it raises the convergence event. In our example, after step 3 going up in the discussion as no sense. Indeed, as the same weight has been attributed to both descriptions of the scanner, B team may choose what it wants for the card reader. Now, suppose that $r_{A,1}$ and $r_{B,2}$ are [(d_scanner, veto), (d'_scanner, 6)], and [(d_card reader, 5), (d'_card reader, veto)], then the discussion would be stopped after step 2 by raising a divergence event.

3.2.2 Coordination protocol

A coordination protocol for a given specification task defines the work steps and their relationships. A work step applies a collaboration type to a set of teams. For example in Figure 3, the work step discussion applies the controlled discussion collaboration type to teams A and B where A has priority over B.

In this section we first give a language to specify relationships between work steps, then we provide a formal definition of the coordination protocol concept.

Coordination expression. A coordination expression is intended to express the relationships between the work steps. It is based on a specific control structure called *coordination iterator.*

A coordination iterator is a loop statement which is controlled by: (1) a classical iteration control expression, (2) exception exit event(s) together with exception handling specification, and (3) execution environment description.

An exception exit event may be a simple event or a composite event built from simple (or composite) events by using some event algebra. The exception handling specification for a given event specifies the action(s) to undertake when the event occurs[1]. The execution environment description specifies a view of the history. A coordination expression is recursively defined by applying the following rules:

R1. a coordination iterator enclosing a work step is a coordination expression,

R2. a coordination iterator enclosing coordination expressions assemblied by either sequence (;) or parallel (|) operators is a coordination expression,

R3. only expressions satisfying rules R1 and R2 are coordination expressions.

Take the controlled discussion example of Section ??, then the underlying coordination expression E would be:

[{(controlled discussion applyied to (A, B)} {iteration number = 3} {convergence => exit,
time_out(1 day)or divergence => abort} {Initial history = $\emptyset$}]

In this expression we have used arbitrary symbols to delimit the coordination iterator ([ ]), and its components ({}). Following, this expression the opinion exchanges betweem teams A and B take place during three iterations, except if an exception exit event occurs. The action to undertake depends on the fact that the exception event is convergence or the composite event "one day time out or divergence". In the first case the execution of the iterator is ended, and in the second case the execution of the entire protocol is aborted. The execution of the expression starts with an empty history.

Formal definition of a coordination protocol. A coordination protocol is a nuplet $< Ct, Te, Ws, Ev, Ce >$, where

- Ct is a set of collaboration types,
- Te is a set of teams,
- Ws is a set of pairs, each of them consisting of a collaboration type of Ct and an ordered set of teams included in Te,

[1]Giving a language for specifying exception handling is out of the scope of this paper, one may envision to use event action rules, or any specification (or programming) language

- Ev is a set of primitive events including exception events associated with collaboration types of Ct,
- Ce is a coordination expression built over Ws and Ev.

The expression E given in the previous paragraph is an example of coordination protocol. Here Ct = {controlled discussion}, Te = {A, B}, Ws contains one element namely <controlled discussion, (A, B)>, Ev = {convergence, divergence, time_out(1 day)}, and Ce is the expression E.

3.3 Protocol execution model

Take the coordination protocol $C1$ above, the control of execution of $C1$ is a standard workflow execution control. Indeed, executing $C1$ can be seen as executing a workflow having two activities A_A and A_B (an activity per team). The end of an activity may be normal (by returning a message) or exceptional (by raising an exception). The controller tests iteration conditions and exception events at the end of each activity. These events may be events raised by activities or external events as for example, time-out. Then the controller decides to start the following activity or to iterate the workflow or to execute actions associated with occurred exception events or to exit the protocol.

Protocol $C1$ is an example of a very simple protocol consisting of only one collaboration iterator, i.e., a protocol built from rule R1. The execution mechanism depicted for $C1$ is generalized to any protocol (i.e., protocols built by using rule R2). The expression E defining a R2 protocol C is an iterator enclosing a set E_1, ..., E_n of coordination expressions assemblied by sequence of parallel operators. Executing C can be seen as executing a worflow having n activities A_1, ..., A_n, where A_i is the activity associated with expression E_i.

Exception handling. The model allows to handle exception events by using standard mechanisms as those proposed in programming languages like for example Java or ADA: at the level of a given iterator, events may be locally processed by using corresponding actions specified in the iterator definition. An event for which no action is defined at a certain level may be processed at an upper level.

Let us revisit the example of Figure 3. A complete coordination protocol P should be of the form:

$$P = [\ \{E\ ;\ F\}\ \{\text{iteration number} = 1\}\ \{\text{decision failure} => \text{action},\ \{\text{Initial history} = \emptyset\}\ ]$$

where E is the coordination expression for the discussion given above and F a coordination expression for the decision. By supposing that a decision collaboration type has been already defined, F should be:

$$F = [\ \{(\text{decision applyied to } (B, A)\}\ \{\text{iteration number} = 1\}\ \{\text{decision failure} => \text{raise decision failure}\}\ \{\text{Initial history} = \text{the most recent messages returned by teams during the execution of } E\ \}\ ]$$

The exception event decision failure may be raised in F, the action undertaken at the F level consists in stopping the execution of F without processing some specific action. The event is handled at P level.

Possible actions undertaken for event exception handling may be: disconnecting from the current protocol, and linking to another one, turning back to some previous work evolution point, stopping the entire protocol, interrupting the protocol execution, or executing some dependent application process.

4 Related Work

This section focuses on the modeling of wokflows and the needs. The needs are well analyzed in [Schmidt, Simone, 1996] and in [Sheth, Kochut, 1997]. In the first paper, the authors define a coordination mechanism as "a construct consisting of a coordinative protocol (an integrated set of pocedures and conventions stipulating the articulation of interdependent distributed activities) on one hand and on the other hand an artifact in which the protocol is objectified". "A coordinative protocol is a resource for situated action in that it reduces the complexity of articulating cooperative work by providing a precomputation of task interdependencies which actors, for all practical purposes, can rely on to reduce the space of possibilities by identifying a valid and yet limited set of options for coordinative action in any given situation". The malleability quality of a workflow model, i.e., its capacity to support dynamic changes, flexible process definitions and graceful handling exceptions, is underlined. In [Sheth and Kochut, 1997], on top of WFMS qualities cited above, the authors stress on the necessity to consider coordination and collaboration. Particularly, the modeling of collaboration and of the relationships between collaboration and coordination tasks are pointed out.

Some researchers adopt advanced transaction models to specify workflows. The traditional concept of transaction has been extended to support long-duration activities and to solve the problem of constraint enforcement in design applications. [Bernstein, Newcomer, 1998] discusses several advanced transaction models, e.g.,: Split-Joint Transactions [Pu, 1988], Flexible Transactions [Elmagarmid & al., 1990], Acta [Chrysanthis, Ramamritham, 1991], Sagas [Garcia-Molina & al., 1991], Contract [Waechter, Reuter, 1992], Open Nested Transactions [Weikum, Schek, 1992].

The main idea of these models is to relax the well-known properties of atomicity, consistency, isolation and durability of the conventional transaction model which would make data unavailable for long time (since transactions may run for days or weeks). Most advanced transaction models allow transactions to be composed of other nested transactions forming a transaction tree. Results of nested transactions are visible to the other sibling transactions or even to external ones. Some models allow the specification of dependencies between transactions. Acceptable states for termination of a transaction in which some sub-transactions may be aborted may be specified by the designer. When concurrency conflicts or failures happen, the compensation concept is used in place of the standard rollback: inner transactions are associated with compensating transactions which leave the database in a consistent state.

Advanced transaction models were developed from a database point of view. Their main concern is the preservation of database consistency. Thus, users and programs are not considered. The disadvantages of this approach were developed in [Alonso & al., 1995] and [Rusinkiewicz, Sheth, 1995].

Other researchers have proposed specific models for workflows. Some of them provide a graphical representation of the flow of activities. This representation is completed by a definition, in a formal language, of the different elements of the workflow: programs used to execute tasks, concerned human actors and related roles, grouping of tasks in activities, start and exit conditions of activities, data structure used, activity execution dependencies and workflow execution failure cases. Among workflow definition languages, let us quote WFDL in [Casati & al., 1995] in which actors and roles concepts do not exist, METEOR in [Sheth & al., 1996], ATREUS in [Grifoni & al., 1997] which provides a lot of possibilities to define activity dependencies, ICN in [Kim, Paik, 1997].

Some other workflow models are based on rules or agents. An example of a rule-based model is provided by [Gokkoda & al, 1997]. Workflows are implemented by using the Acta transaction model and the rule specification language provides Acta-like facilities to declare task dependencies and compensated activities.

[Leymann, Roller, 1998] and [Divitini & al., 1996] propose agent-based models. Both papers concentrate on agent communications. But, the first paper focuses on the implementation of agent communications while the second one is concerned with the specification of communications. In more details, [Leymann, Roller, 1998] describes a conceptual workflow architecture based on persistent queues. Each agent has an input and an output queues. Communications between agents are realized by messages put in the agent input queues. Messages are deleted from the input queues by its consumer. The results of agent actions are messages inserted by the agents in their output queue.

[Divitini & al., 1996] presents a language, IL, which allows to define the interoperability between agents in a multi-agent-based CSCW systems. IL integrates the malleability and linkability qualities of computational coordination mechanisms. In particular, special working modes, like conversation (what we called discussion in our examples), awareness and iteration are treated as independent and linked coordination mechanisms. IL provides low-level means to define communications between the different agents of a coordination mechanism (awareness with or without acknowledgement of receipt, sending of information, task activation, warning) and between different coordination mechanisms. IL allows to define conditions associated with the communications. The real executed protocol will depend on the satisfaction of these conditions (malleability quality).

Another idea is provided by [Ailamaki & al.,1998]: workflows are represented as schemas of an object-oriented database. Two kinds of classes exist: one for data used or produced by the tasks and one for tasks themselves. Relationships connect each task to the data it uses and to the data it produces. The execution of tasks are controlled by triggers on the relationships.

5 Conclusion

In this paper we describe a framework for specifying cooperative work integrating both collaboration and coordination between partners. The both aspects are modeled in an unified way in the form of a workflow where, instead of considering the workflow tasks as black boxes, we consider that the tasks (called work step in our model) may con-

cern several teams having to collaborate. So the workflow specification is enriched by the description of the collaboration aspect which is present inside the tasks. We provide a declarative language for specifying what we call a coordination protocol from existing ones: by adapting one worflow, or by assembling several workflows in a serial or parallel manner leading to nested workflows. The language allows to define exception events, to specify circumstances where the events may be raised, what actions have to be undertaken when exception events occur, and at what point in the workflow these actions have to be executed. The main advantages of such a declarative approach are to provide collaborative workflows which are easy to understand, thus easy to re-use and to adapt. This approach is particularly well-suited in the context of large-scale project designing which involves different engineering and management expertise. In such a context, collaborative workflows may be kept in the corporate memory to be further used by new members who want to learn the work done in the past. In our framework, the generic collaboration type may easily contribute as a constituant of the corporate memory. Providing generic specification of coordination protocols is actually an open issue that we envision to study in a near future. Other works we are interested on, deal with the implementation of our model, in particular the algorithmic aspect of the history management.

References

A. Ailamaki, Y. E. Ioannidis, M. Livny, 1998, *Scientific Workflow Management by Database Management*, in Proceedings of 10th Int. Conf. on Scientific and Statistical Database Management, July 1-3, Capri, Italy

G. Alonso, D. Agrawal, A. El Abbadi, M. Kamath, R. Günthör, C. Mohan, 1995, *Advanced Transaction Models in Workflow Contexts*, Research Report RJ 9970 (87929) 7/31/95, IMB Research Division

P. Bernstein, E. Newcomer, 1998, *Principles of Transaction Processing*, Morgan Kaufmann Publishers

MJ. Blin, F. Fabret, 1999, A Coordination Mechanism to Prevent the Violation of Distributed Constraints, Cahiers du Lamsade no. 164, Universit Paris-Dauphine

F. Casati, S. Ceri, B. Pernici, G. Possi, 1995, *Conceptual Modeling of Workflows*, in Proceedings of the 14th International Conference on Object-Oriented and Entity-Relationship Approach, Australia, Springer-Verlag, Lecture Notes in Computer Science

P. Chrysanthis, K. Ramamritham, 1991, *A Formalism for Extended Transaction Model*, Proceedings of the Seventeenth International Conference on Very Large Data Bases.

A. A. Clarke, 1996, *A Theoretical Model of Cooperation*, Proceedings of COOP'96, June 12-24, Juan-les-Pins, France

M. Divitini, C. Simone, K. Schmidt, 1996, *Abaco: Coordination Mechanisms in a Multi-agent Perspective*, Proceedings of COOP'96, June 12-24, Juan-les-Pins, France

A. K. Elmagarmid, Y. Leu, W. Litwin, M. Rusinkiewicz, 1990, *A Multidatabase Transaction Model for Interbase*, Proceedings of the Sixteenth International Conference on Very Large Data Bases.

H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, K. Salem, 1991, *Modeling Long-Running Activities as Nested Sagas*, In Proceedings IEEE Spring Compcon

E. Gokkoda, M. Altinel, I. Cingil, E. N. Tatbul, P. Koksal, A. Dogac, 1997, *Design and Implementation of a Distributed Workflow Enactment Service*, in Proceedings of Second IFCIS International Conference on Cooperative Information Systems, Kiawah Island, South Carolina, USA

P. Grifoni, D. Luzi, P. Merialdo, F. L. Ricci, 1997, *Atreus : a Model for the Conceptual Representation of a Workflow*, in Proceedings of the Eighth International Workshop on Database and Expert Systems Applications, September 1-2, Toulouse, France

S. Jablonski, C. Bubler, 1996, *Workflow Management: Modeling Concepts, Architecture and Implementation*, International Thomson Computer Press

K.-H. Kim, S.-K. Paik, 1997, *Practical Experiences and Requirements on Workflow*, in Coordination Technology for Collaborative Applications, W. Conen and G. Neumann, editors, Springer-Verlag

F. Leymann, D. Roller, 1998, *Building a Robust Workflow Management System with Persistent Queues and Stored Procedures*, in Proceedings of the Fourteenth International Conference on Data Engineering, February 23-27, Orlando, Florida

B. Munier, M. F. Shakun, 1988, *Compromise, Negotiation and Group Decision*, D. Reidel Publishing Compagny

C. Pu, 1988, *Superdatabases for Composition of Heterogeneous Databases*, IEEE Proceedings of The Fourth Conference on Very Large Data Bases

M. Rusinkiewicz, A. Sheth, 1995, *Specification and Execution of Transactional Workflows*, in Modern Database Systems, Won Kim, editor, Addison Wesley

K. Schmidt, C. Simone, 1996, *Coordination Mechanisms: Towards a Conceptual Foundation of CSCW Systems Design*, JCSCW, vol. 5, no. 2-3

A. Sheth, K. Kochut, 1997, *Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems*, in Proceedings of the NATO ASI on Workflow Management Systems and Interoperability, Aug, Istanbul, Turkey

A. Sheth, K. Kochut, J. Miller, D. Palaniswami, J. Lynch, D. Worah, S. Das, C. Lin, I. Shevchenko, 1996, *Supporting State-wide Immunization Tracking using Multi-Paradigm Workflow Technology*, in Proceedings of the 22nd International Conference on Very Large Data Bases, Mumbai(Bombay), India

H. Waechter, A. Reuter, 1992, *The ConTract Model*, in Database Transaction Models for Advanced Applications, A. K. Elmagarmid, editor, Morgan Kaufmann Publishers

G. Weikum, H.-J.Schek, 1992, *Concepts and Applications of Multilevel Transactions and Open Nested Transactions*, in Transactions Models for Advanced Applications, A. K. Elmagarmid, editor, Morgan Kaufmann Publishers